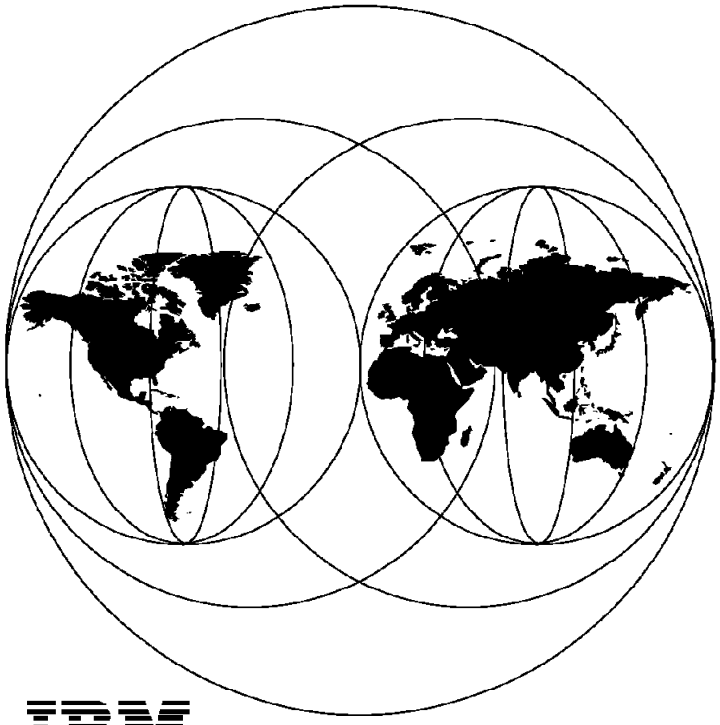
SG24-4630-00

International Technical Support Organization

OS/2 Warp (PowerPC Edition)
A First Look

December 1995



IBM
International Technical Support Organization
Boca Raton Center

SG24-4630-00

International Technical Support Organization

OS/2 Warp (PowerPC Edition)
A First Look

December 1995

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xiii.

First Edition (December 1995)

This edition applies to OS/2 Warp Connect (PowerPC Edition) Version 1.0.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. JLPC Building 14 Internal Zip 5220
1000 NW 51st Street
Boca Raton, Florida 33431-1328

Abstract

This document is unique in its detailed coverage of OS/2 Warp (PowerPC Edition) architecture. It focuses on the architecture of the microkernel. It provides information about the microkernel based architecture, which will give dramatic advances in computing architecture.

This document was written for IBM customers, system engineers, software developers. Basic understanding of OS/2 is assumed.

(178 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help IBM customers, dealers and IBM system engineers to provide an introduction in the architecture for OS/2 Warp (PowerPC Edition). The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/2 for PowerPC. See the PUBLICATIONS section of the IBM Programming Announcement for OS/2 for PowerPC for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

FFST First Failure Support Technology
IBM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Windows is a trademark of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.

Preface

This document is intended to provide an overview about the OS/2 Warp Connect (PowerPC Edition) architecture. It contains information about the microkernel which is the base for OS/2 Warp Connect (PowerPC Edition) architecture.

This document is intended for IBM customers and employees requiring an overview of the OS/2 Warp Connect (PowerPC Edition) architecture.

How This Document is Organized

The document is organized as follows:

- Chapter 1, “Introduction”

  This chapter will give you an introduction to OS/2 for PowerPC.

- Chapter 2, “The IBM Microkernel”

  This chapter describes the IBM microkernel. It will give an overview of the new foundation for operating systems.

- Chapter 3, “System Services”

  This chapter provides an overview of the system services, which behave as a server to the operating system.

- Chapter 4, “OS/2 Functions”

  This chapter provides an overview of the reasons why the OS/2 Warp (PowerPC Edition) was conceived, and it also gives you an overview of the OS/2 Warp (PowerPC Edition) architecture.

- Chapter 5, “Installation”

  This chapter describes installation process of OS/2 Warp (PowerPC Edition).

- Chapter 6, “Application Support”

  This chapter describes application support and tools to convert from Intel platform to PowerPC platform.

- Appendix A, “Changes to MVM DOS Settings”

  This appendix describes the changes to the MVM DOS settings in OS/2 Warp (PowerPC Edition) versus the OS/2 Warp (Intel version).

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *Inside the PowerPC Revolution (by Jeff Duntemann and Ron Pronk)* , ISBN 1-883577-04-7
- *Exploring the PowerPC Revolution*, ISBN 1-885068-02-6

International Technical Support Organization Publications

- *OS/2 Version 2.0. Volume1: Control Program*, GG24-3730
- *OS/2 Warp (PowerPC Edition) Graphical Subsystem* , SG24-4639
- *PowerPC Series*, SG24-4592 (not available yet)

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Visa and MasterCard are accepted. Outside the USA, customers should contact their local IBM office. Guidance may be obtained by sending a PROFS note to BOOKSHOP at DKIBMVM1 or E-mail to bookshop@dk.ibm.com.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser to the following URL:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. The internal redbooks home page may be found at the following URL:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Acknowledgments

This project was designed and managed by:

Lajos Damen
International Technical Support Organization, Boca Raton Center

Alex Gregor
International Technical Support Organization, Boca Raton Center

The authors of this document are:

Joachim Birke
IBM Germany

Rudi van Emmenes
IBM South-Africa

Juliandri Jenie
IBM Indonesia

Scott Rigby
IBM Australia

Terje Storstein
IBM Norway

This publication is the result of a residency conducted at the International Technical Support Organization, Boca Raton Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Scott Bennett
IBM OS/2 Warp (PowerPC Edition) Installation development, Boca Raton

Craig A. Bennett
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Kenneth W. Borgendale
IBM OS/2 Warp (PowerPC Edition) architect, Boca Raton

Arnold H. Bramnick
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Mike Cooper
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Robyn L. Focazio
IBM Microkernel development, Austin

Steve C. Heuer
IBM Microkernel development, Austin

Bryon S. Neitzel
IBM OS/2 Warp (PowerPC Edition) MVM development, Boca Raton

Chris P. Perritt
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Pete C. Rodriguez
IBM OS/2 Warp (PowerPC Edition) Installation development, Boca Raton

James R. Schoech
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Jim White
IBM Boca Raton

Kenneth E. White
IBM OS/2 Warp (PowerPC Edition) development, Boca Raton

Chapter 1. Introduction

OS/2 is now, for the first time, running on a different platform from the Intel based personal computer. It now also runs on systems based on the PowerPC architecture defined by the Apple, IBM and Motorola alliance. The implementation of OS/2 Warp Connect (PowerPC Edition) is based on the IBM microkernel.

The IBM microkernel is the foundation for a set of highly portable systems. The microkernel provides a way of structuring systems software to reduce complexity and increase its flexibility and portability.

The microkernel contains a small, message passing nucleus of system software running in the most privileged state of the computer that controls the basic operation of the machine. The IBM microkernel includes the microkernel and a set of servers and device drivers that provide microkernel services.

The functions performed by the microkernel are limited in order to reduce its size and to maximize the amount of code that runs outside the kernel. The microkernel includes only those functions required to provide a set of abstract processing environments for application programs and to permit applications to work together to provide services and acts as clients and servers.

Many other operating system functions, such as file systems, device support, and traditional programming interfaces are placed outside of the microkernel for possible reuse of other operating systems developers.

The IBM microkernel uses technology from the Carnegie Mellon University MACH Research Project. The IBM microkernel also incorporates selected technology developed by the Open Software Foundations Research Institute.

This book gives you a first look in the components of OS/2 Warp Connect (PowerPC Edition) as depicted in Figure 1 on page 2.

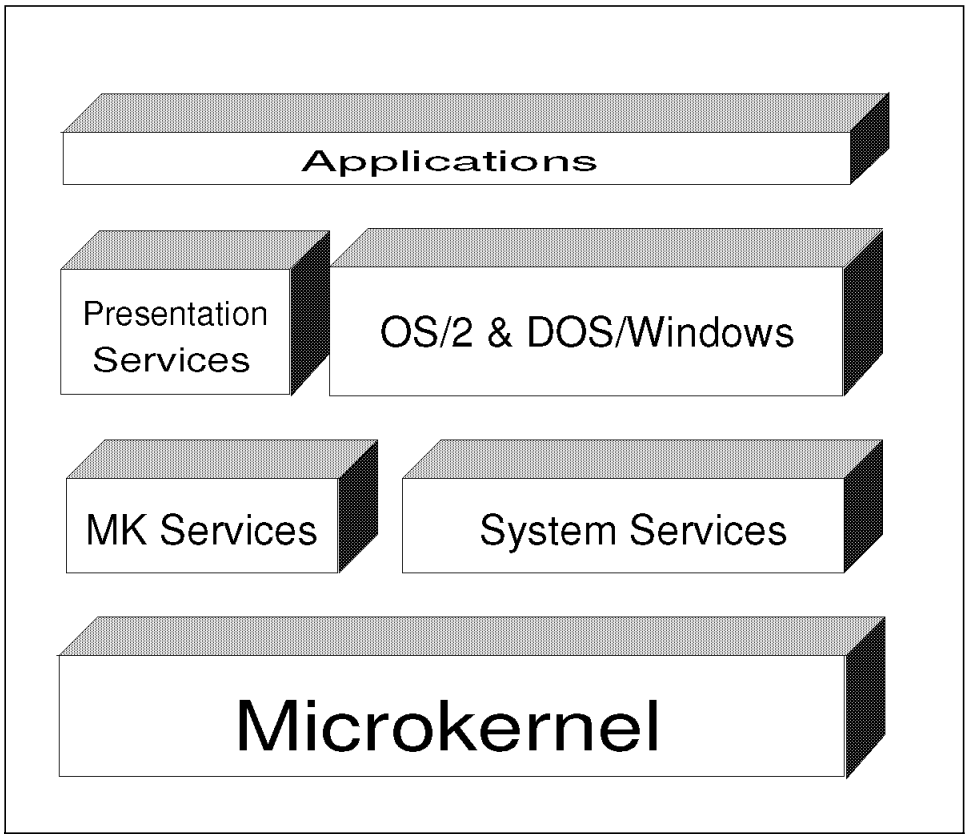


Figure 1. OS/2 Warp Connect (PowerPC Edition) Components

The IBM microkernel and the microkernel services are described in Chapter 2, “The IBM Microkernel” on page 5.
The system services depicted in Figure 1 consist of the services which might be useful to other operating system developers using the microkernel as a base. The system services are:

- Event and window services
  The main part of the session management of OS/2 is handled by this component. Event and window services are also responsible for handling all keyboard and mouse (and any other locator device) input.
- File services
  Are responsible for handling all the file requests in OS/2 Warp Connect (PowerPC Edition).

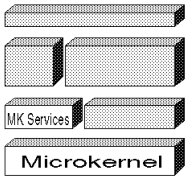
- Pipe services
  All the pipe requests presented by application programs to the DOSCALLS.DLL API library are managed by the pipe server.

The system services are described in Chapter 3, “System Services” on page 31.
Chapter 4, “OS/2 Functions” on page 51 describes the OS/2 Control Program and the Dos/Windows support of OS/2 Warp Connect (PowerPC Edition).
The presentation services, which consist of the user interface, and the components necessary to support it, are the topics of 4.3, “Graphics Subsystem” on page 87.

To use the system, it has to be installed on a PowerPC-based machine. This process is described in Chapter 5, “Installation” on page 129.

Applications supported, application migration considerations and applications development, are the topics of Chapter 6, “Application Support” on page 141.

Chapter 2. The IBM Microkernel



The OS/2 Warp Connect (PowerPC Edition) product is based on the microkernel technology as shown in Figure 1 on page 2.

The idea of a microkernel-based operating system dates back to 1986, when a team of researchers at Carnegie-Mellon University addressed several UNIX problems, mainly concerning the inability to extend the UNIX operating system due to its layered structure and the difficulty to easily port an application from one vendor specific UNIX version to another vendor specific UNIX version. The team at Carnegie-Mellon University solved this problem by designing a client/server and message based microkernel, called Mach. The IBM microkernel is based on this MACH microkernel and it also incorporates selected technology developed by the Open Software Foundation Research Institute. Additionally, IBM has made several improvements to the microkernel in order to get a solid base for the OS/2 Warp Connect (PowerPC Edition) product.

The IBM microkernel provides the following comprehensive environment for operating systems:

- Multiprocessing
- Multithreading
- Interprocess communication
- Extensible memory management
- Support for multiple operating personalities

The IBM microkernel also provides a concise set of IBM microkernel services implemented as a pure kernel and an extensive set of services for building operating system personalities implemented as a set of user-level servers.

Functions of the IBM microkernel include the following:

- Providing common programming for low-level system elements, such as device drivers and file systems.
- Exploiting parallelism in both operating system and user applications.
- Supporting large address spaces with flexible memory sharing.
- Allowing transparent network resource access.
- Enabling compatibility with existing software environments, such as OS/2 and DOS/Windows.
- Enabling portability (to 32-bit and 64-bit platforms).

2.1 Elements of the IBM Microkernel

The IBM microkernel provides the following small set of abstractions:

- **Task** Unit of resource allocation: large address space and port right
- **Thread** Unit of CPU utilization: lightweight (low overhead)
- **Port** A communication channel accessible only through the send/receive capabilities or rights
- **Memory object** The internal unit of memory management

The functions performed by the microkernel are limited in order to reduce its size and maximize the amount of code that runs outside the kernel. The microkernel includes only those functions required to provide a set of abstract processing environments for application programs and to permit applications to work together to provide services and act as clients and servers. The five type of services are:

1. Physical resource management
2. I/O support and interrupt management
3. Interprocess communication (IPC)
4. Tasks and threads
5. Virtual memory management

These services offered by the IBM microkernel will be described in more detail in the following paragraphs.

2.1.1 Physical Resource Management

The IBM microkernel brings the various resources it maintains to virtual memory. However, all actions performed depend on the underlying physical resources of the IBM microkernel.

Host Machines

A host is the multiprocessor as a whole. Each host (uniprocessor or multiprocessor) in a networked IBM microkernel system runs its own instantiation of the IBM microkernel. The host multiprocessor is not generally manipulated by client tasks. But, because each host does carry its own IBM microkernel, each with its own port space, physical memory, and other resources, the executing host is visible and sometimes manipulated directly. Also, each host generates its own statistics.

Hosts are named by a name port, which is freely distributed and can be used to obtain information about the host, and a control port, which is closely held and can be used to manipulate the host. Operations supported by hosts include the following:

- Clock manipulation
- Statistics gathering
- System reboot
- Setting the default memory manager
- Obtaining lists of processors and processor sets
- Accessing hardware

Physical Processors

A physical processor is a unit capable of executing threads. Each physical processor is named by a processor control port. Although significant in that they perform the real work, processors are not very significant in the microkernel, other than as members of a processor set. It is a processor set that performs the basis for the pool of processors that is used to schedule a set of threads and has scheduling attributes associated with it.

Processor Sets

Processors are grouped into processor sets. A processor belongs to only one processor set. A processor set forms a pool of processors used to schedule the threads assigned to that processor set. A processor set exists

as a basis to uniformly control the schedulability of a set of threads. The notion also provides a way to perform coarse allocation of processors to given activities in the system.

A host contains a number of processor sets, including a default processor set.

The operations supported upon processor sets include the following:

- Creation and deletion
- Assignment of processors
- Assignment of threads and tasks
- Scheduling control

Clocks

A clock provides a representation of the passage of time by incrementing a time value counter at a constant frequency. Each host or node in a multicomputer implements its own set of clocks based upon the various clocks and timers supported by the hardware as well as abstract clocks built upon these timers. The set of clocks implemented by a given system is set at configuration time.

Each clock is named by both a name and a control or privileged port. The control port allows the time and resolution of the clock to be set. Given the name port, a task can perform the following:

- Determine the time and resolution of the clock
- Generate a memory object that maps the time value
- Sleep (delay) until a given time
- Request a notification or alarm at a given time

The clock facility implements one or more of the following clocks:

- The REALTIME clock which measures with moderate resolution the time since system initialization.
- The standard defined BATTERY clock with low resolution which provides a time value that is controlled exclusively by user-level code.
- The standard defined HIGHRES clock which enables high-resolution alarm services.

Physical Memory

The various IBM microkernel objects and associated data structures occupy physical memory. It is a hardware- and implementation-dependent issue as to which of these structures can be swapped or paged out of memory. Currently, clients have no control over this memory, except to the extent that they create kernel-managed entities.

The majority of the system′s physical memory forms a single paging pool. The pool of pages forms a cache for the virtual memory system. The set of pages that resides in physical memory at any given time is decided by the page-replacement algorithm, implemented in the kernel. Clients have no control over this algorithm, with the exception of the vm_wire call which forces a region of virtual memory to be and stay resident. Even external memory managers have no influence; if they do not respond fast enough to a request to write a page, the default memory manager is used to move the page from physical memory to system paging storage.

When a memory object is no longer referenced, the kernel normally frees all of its physical memory pages. A memory manager can mark an object′s pages as cacheable, meaning that the object′s pages are moved to a free list but are not destroyed. This is usually specified for memory objects that persist.

2.1.2 I/O Support

A modern computer system may support a wide range of buses, controllers, and devices. The configuration services are responsible for locating and managing all of the I/O related hardware resources that are visible to the software in the system. They determine what I/O hardware is present and grant device code access to it.

These configuration services consist of:

- The configuration manager, which identifies the machine and any built-in hardware.
- The device manager, which is responsible for starting the device drivers.
- The resource manager, which controls this process.

In order to support I/O and device access, the microkernel provides access to I/O resources, such as memory-mapped devices, I/O ports, and direct memory access (DMA) channels. The DMA interfaces are used in providing information and in programming certain hardware functions to transfer data between memory and a device.

2.1.3 Inter Process Communication (IPC)

The majority of interactions between an IBM microkernel task and its environment are accomplished by sending and receiving messages. To facilitate this, the microkernel provides synchronous one-way messaging as well as a Remote Procedure Call (RPC) mechanism. Regardless of the mechanism employed, all forms of IPC occur over IBM microkernel-provided communication channels known as ports.

Ports

A port is a unidirectional communication channel between a client that requests a service and a server that provides the service. A port has a single receiver (server) and potentially multiple senders (clients) which are connected in a secure fashion.

With the exception of the task′s virtual address space, all other system resources are accessed through ports.

Any given entity can have multiple ports that represent it, each implying different sets of permissable operations. For example, many entities have a name port and a control port that is sometimes called the privileged port. Access to the control port allows the entity to be manipulated. Access to the name port simply names the entity, for example, to return information.

There is no system-wide name space for ports. A thread can access only the ports known to its containing task (port name space, see below). A task holds a set of port rights, each of which names a (not necessarily distinct) port and which specifies the rights permitted for that port.

Every port is created as an instance of a port class. When created, a single receive right (creator is server) for the port is established and also added to the specified port name space.

Port Name Space (Portspace)

Port rights (port names) cannot be accessed directly but, instead, are accumulated in port name spaces. The port name space assigns a local name, for the right, that is used to access the right. Each task is associated with a port name space that provides the context for interpreting names into rights for all threads.

The names assigned within a port name space are completely at the discretion of the Inter Process Communication (IPC) system. The user has no control over these names, but the following guarantees are made:

- Receive and send (including the send-once restricted form) rights to the same port coalesce to a single port name. That is, if a port name space holds three send rights and a single receive right for a port, it will have a single name for all four rights.
- Send rights are reference counted but only a single receive right can exist for a port or port set.
- Port names are freed when all the reference counts go to zero.

Because a port name space is bound to its owning task, it is created and destroyed with its owning task.

Port Classes

A port class defines the format of messages that can be transferred across ports of this class. A port class may also be created as a combination of the signature of a base port class and a specified new signature (signatures are described in the messages section). Most ports have well defined messages that are passed across them. This approach allows these formats to be preregistered with the IPC system once, avoiding the overhead of verifying their correctness on each message transfer.

After the port class is created, its message format can never change. Additionally, the port class associated with a port cannot change and it also cannot be explicitly destroyed. Instead, the port class will be retained until the last reference to the port class goes away.

Port Rights

Because of their fundamental nature in the workings of the microkernel system, ports are strongly protected. A port can be accessed only according to the set of capabilities granted by the user. These capabilities, known as port rights, are maintained on a port name space basis. Each task has an associated port name space, and therefore can have a unique combination of port rights for a particular port. This capability notion is the fundamental security model and mechanism exported by the IBM microkernel.

The following port rights are maintained by the Inter Process Communication (IPC) system:

- **Receive right**

  This capability enables the holder to receive messages from the port (holder acts as a server). Only a single receive right exists for a port, and after it is destroyed it cannot be recreated. Therefore, a port whose

receive right has been destroyed is considered dead. The receive right also gives the holder the right to make send rights for the port.

- **Send right**

  This capability enables the holder to send unlimited messages over the port. Many send rights (clients) can exist for a single port.

- **Send-once right**

  This capability enables the holder to send a single message over a port. Many send-once rights can exist for a single port.

Notifications

Many tasks using a port can be notified, through a kernel-generated Remote Procedure Call (RPC), when certain state transitions occur relative to the port. Such notifications are requested when one of the following conditions occur:

- **Dead-Name state**

  When the receive right for a port is destroyed (server does not exist anymore) this port is considered to have died. Any task which holds a send right for this port can be notified of this state transition, when a registration has been acquired accordingly.

- **No-More-Senders state**

  When the last send or send-once right for a port is deallocated, the port is equally unusable for message transmission. The holder of the receive right for the port might be interested in this event in order to perform garbage collection of resources associated with the port. Such tasks can register for no-more-senders notifications.

Port Sets

A port set is a means of collecting a number of receive rights together into a single unit for message-receipt purposes. When a receive operation is performed against a port set, a message will be received at random from one of the ports in the set. It is not allowed to directly receive a message from a port that is a member of a port set. There is no notion of priority for the ports in a port set.

The port set has its own name. A receive right can belong to only one port set.

Messages

A message is a structured collection of direct data, indirect data, and port rights passed between two entities through transmission over a port. The message itself consists solely of data, addresses and port right names. Its contents are interpreted based upon the signature registered for the port in order to facilitate the data transfer.

For each particular port there may exist several message IDs and for each message ID there is one signature which describes the elements of this message. Thus, all signatures belonging to a particular port are described in a data structure called signature collection, which has the following contents:

- The size, in bytes, of the signature collection
- A contiguous range of message IDs that are valid on the port
- An array of offsets into the signature collection for each message ID′s signature

Message Transmission

The fundamental microkernel mechanism for message transmission is a form of two-way messaging most closely related to Remote Procedure Call (RPC) implementations. However, if the signature for a particular message ID defines no return data, the client may choose to send the message as a one-way message by invoking a special interface.

The following applies when messages are sent:

- The interpretation and transmission of the supplied message buffer is driven by the message signature.
- All memory-addressing errors are handled through exceptions, not by returning errors. The offending task will be terminated, unless the exception is handled.
- All other errors are reported synchronously through return codes from the messages APIs.
- Resource shortages in the sender, receiver, or kernel will cause message transmission be silently blocked until resources become available.

As already mentioned, there are two type of interprocess communication within the microkernel environment: the Remote Procedure Call (RPC) and the one-way message.

Remote Procedure Call (RPC)

Although it is possible to emulate the RPC at user level through the use of two ports and the one-way message-passing, such an approach could not equal the performance offered by a kernel-supplied primitive tuned for RPC. With this approach, a sender supplied reply port is not needed as the microkernel RPC maintains a reply port inside the kernel, thus avoiding overhead. Additional overhead at user-level code is avoided because verifying messages on reply is not necessary because the reply message format comes from the same signature.

Figure 2 shows the kernel-supplied reply ports and the RPC linkage between the client and the server.

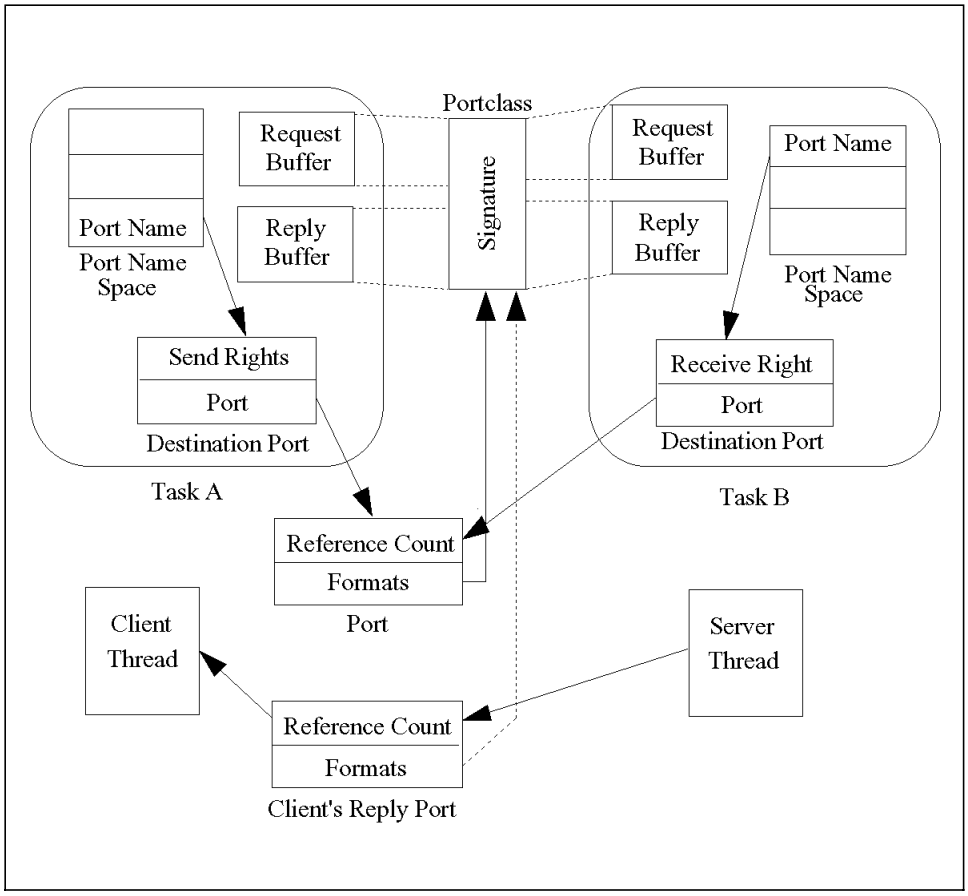


Figure 2. RPC Linkage between Client and Server

One-Way Inter Process Communication

In a one-way IPC, a message is sent without expecting a reply. The signature for the corresponding message ID must specify that no reply data is required. Unlike RPC, as soon as the data transfer to the server is complete, the corresponding call completes.

Server Thread Support

A common practice is to have a set of threads in a task dedicated to receiving messages and generating replies (where appropriate). This is the sole function of these threads. Because this approach is so common, the microkernel provides direct support for this practice.

After a thread has been made a message server, it cannot be safely removed from its task without running the risks associated with nonrestartable aborts.

Message Interface Generator

The Message Interface Generator is a program that generates Remote Procedure Call (RPC) code for communication between a client and a server process.

To use the Message Interface Generator the user has to provide a specification file which defines the parameters for the message passing interface and the procedure call interface.

Due to the contents of the specification file the Message Interface Generator generates three files:

- User (Client) Interface Module: It implements and exports procedures and functions to send and receive the appropriate messages to and from the server.
- User (Client) Header Module: It defines the types and routines needed at compilation time.
- Server Interface Module: It extracts the input parameters from an IPC message and calls a server procedure to perform the operation. When this procedure returns, the generated interface module returns the procedures return code in the reply message with all output parameters sent by reference. Note that this generated module does not perform the action of receiving or sending messaging, only the interpretation and processing of messages. Instead, the Message Interface Generator

provides a ″demultiplexer (demux)″ function, which, after having interpreted the incoming message, calls the desired function to perform the actual work.

2.1.4 Tasks and Threads

The IBM microkernel architecture defines tasks and threads in order to support parallel execution. This is done by separating the execution environment (tasks) from the execution of instruction streams (threads). That means, a task does not execute itself. Threads are the active and computational entities. So, by saying ″task A does X″, it is meant, that ″a thread contained within task A does X″.

2.1.4.1 Tasks

A task is a container to hold references to resources in the form of:

- A port name space
- A virtual address space
- A set of threads

All tasks are tagged with a security token, an identifier that is opaque from the IBM microkernel′s point of view. It encodes the identity and other security attributes of the task. This security token is included as an implicit value in all messages sent by the task.

A new task is created based on an existing prototype task. The new task:

- Has either an empty virtual address space or one inherited from the parent task.
- Inherits the security token from its parent task.
- Has an empty port name space.
- Contains no threads.

The new (child) task receives the following special ports, which are created or copied at task creation:

- **Task-self port**

  The port by which the kernel knows the new child task and allows it to be manipulated. The child task holds a send right for this port. This port name is also returned to the calling task.

- **Bootstrap port**

  The port to which the child port can send a message requesting return of any system service port it needs. The child task inherits a send right for this port from the parent task.

- **Host-self port**

  The port by which the child task requests information about its host. The child task inherits a send right for this port from the parent task.

Priority

As threads are the only computational entities within a task, the priority of a task has only an effect on containing tasks, that is the priority of a new thread is set to match the priority of the enclosing task.

2.1.4.2 Threads

As already mentioned, threads are the basic computational, active entities in the IBM microkernel. A thread is a lightweight entity which is inexpensive to create and requires low overhead to operate. Its owning task bears the burden of resource management. On a multiprocessor, it is possible for multiple threads in a task to execute in parallel. A thread belongs to only one task that defines its virtual address space and a port name space with which other resources are accessed.

A thread has the following features:

- A point-of-control flow in a task or a stream-of-instruction execution.
- Access to all the elements of the containing task.
- Parallel execution with other threads, even threads within the same task.
- Minimal state for low overhead.

A thread has the following set of states:

- **Machine state**

  It changes as the thread executes and can also be changed by another holder of the corresponding IBM microkernel thread port. But care should be taken to set the state of the thread because inconsistency may occur.

- **A set of thread-specific port rights**

  This set identifies the thread′s microkernel port, a reply port maintained for the thread by the kernel, and ports used to send exception messages on behalf of the thread.

- **Suspend count**

  Is nonzero if the thread is not to execute instructions.

- **Resource scheduling parameters**

  For example, assignment to a specific processor set or scheduling policy for a corresponding processor set.

- **Thread security token**

  Provides a thread override to the task proxy security token.

Priority and Scheduling

A thread is scheduled for execution according to its current priority and the scheduling policy currently set for the thread's assigned processor set. There are scheduling policies defined, such as:

- POLICY_TIMESHARE
- POLICY_RR (round robin)
- POLICY_FIFO (first-in, first-out)

Threads have three priorities associated with them by the system:

- A priority value that can be set by the thread to any value up to a maximum priority.
- A maximum priority value that can be raised only through privileged operation so that users cannot unfairly compete with other users in their processor set.
- A scheduled priority value used to make scheduling decisions for the thread. This value is determined on the basis of the user priority value by the scheduling policy.

Processor Sets

As already mentioned, tasks are assigned to a specific processor set. When a new thread is created in that task, this thread inherits the corresponding processor set. However, a thread can be assigned a different processor set.

Traps and Exception Processing

To affect the structure of the address space or to reference any resource other than the address space, the thread must execute a special trap instruction. This causes the IBM microkernel to perform operations on behalf

of the thread or to send a message to an agent on behalf of the thread. These traps manipulate resources associated with the task containing the thread.

Scheduling Support Traps

Normally, threads are preemptively scheduled by the microkernel according to its scheduling policies. When a thread wants to give up the processor it can do so by issuing the thread_switch trap by specifying another thread to run.

Another scheduling trap is the clock_sleep trap, which delays the invoking thread until a specified time.

Identity Traps

These traps are used by the thread in order to initially obtain the port rights for itself and its task.

Message Send and Receive Traps

The most important set of the IBM microkernel traps are those used to send and receive messages or make and service Remote Procedure Call (RPC).

Exception processing

When an exception occurs in a thread, the thread executes in kernel context and sends a message whose contents describe the exception to an exception port. For any given exception, two exception ports apply:

- A thread-specific type of exception
- A task port for the specific type of exception

The type of exceptions for which the exception ports applies are, for example:

- Arithmetic exception
- Could not access memory
- Invalid or undefined instruction or operand
- Software generated exception

After an exception, the kernel selects the thread-specific port for the specific type of exception as the destination or the exception message (if it is defined). Whereas a successful reply causes the thread to continue, an

unsuccessful reply causes the kernel to send an exception message to the task port for the specific exception. If neither exception message receives a successful reply, the thread is terminated.

Not every exceptional condition that a thread encounters is handled this way. A page-not-resident does not send a message to the exception port. Instead, a message is sent to the external memory manager associated with the memory page in which the faulting address lies.

Creating Kernel Threads and Tasks

For the sake of its own operation, the kernel creates kernel threads that execute purely within kernel context to provide various support functions. For example, page-out function, thread reclamation, and scheduler priority computations are performed by dedicated threads, rather than being executed in interrupt or software interrupt context. Users of the system, including privileged ones, have no direct control over these internal threads. To provide a task context for these threads, the kernel constructs a kernel task to contain them.

2.1.4.3 C-Threads

The IBM microkernel provides a set of low-level, language-independent primitives for manipulating kernel-level threads of control in support of multithreaded programming as described in the foregoing sections. Additionally, there is a C-Threads package, which is a run-time library that provides user-level threading as well as a C language interface to these facilities. The constructs provided are as follows:

- Forking and joining of threads
- Protection of critical regions with mutex variables
- Synchronization by means of condition variables

A set of C threads can execute in parallel on multiple processors within a system. There is a one-to-one mapping of C threads to kernel level threads.

2.1.5 Virtual Memory Management

In order to describe the basic mechanisms of virtual memory management, the following elements will be defined:

Memory object

A memory object is an abstract image of a set of ordered bytes. All data in the system is represented by memory objects. Memory objects can be

referenced by mapping portions of the memory object into a range of virtual addresses in the virtual address space.

Memory Manager

A memory manager is a microkernel task that maintains a set of memory objects. For example, a file in the file system could be represented as a memory object in order to provide memory mapped access to that file. This memory manager may be the pager component of a file system that maintains the abstract image of the memory object on a permanent backing media.

Virtual address space

A virtual address space consists of a range of virtual addresses, beginning at a minimum virtual address and extending to a maximum virtual address. The virtual address space is divided into virtual pages.

Virtual pages are cached in physical memory. As in all virtual memory systems, the total size of all the memory object currently being referenced can be greater than the amount of physical memory in the system.

Creating Virtual Address Spaces

A virtual address space is created when a task is created and destroyed when the task is destroyed. Normally the new task inherits the virtual address space of its parent task, for example it acquires shared or copied parts of the parent′s address space. Alternatively, a child task can be created with an empty address space.

Allocating Virtual Memory

Allocating virtual memory can be done in two ways which results in either:

- A mapping of a portion of a memory object into the virtual address space
- A range of memory initialized with zeros (anonymous memory)

Working with Virtual Memory

There are functions provided by the kernel to copy virtual memory from one task to a different one as well as within the own virtual address space. In order to prevent random allocation of virtual memory within a specific region of the virtual address space, this region might be reserved.

Setting the Protection/Inheritance Attribute

Access permission for a specific virtual address region are:

- **VM_PROT_NONE**
- **VM_PROT_READ**
- **VM_PROT_WRITE**
- **VM_PROT_EXECUTE**
- Any combination thereof

Note that enforcement of protection attributes as well as combinations are machine-dependent. This is also valid for the semantics of attributes or combinations of attributes. For example, for some hardware platforms write access implies read access and execute access cannot be distinguished from read access.

Each virtual address region has a maximum and a current protection. The current protection must be a subset of the maximum protection. The maximum protection cannot be changed to include additional protection accesses.

Using Virtual Address 0

Some programs fail if memory is allocated at address 0, because they consider a memory pointer whose value is 0 to be a null pointer and not a pointer to a valid memory byte. But some functions may allocate a region at address 0. To prevent the kernel from doing this, the first page at address 0 should be reserved.

2.2 Elements of the IBM Microkernel Services

The microkernel services portion of the IBM microkernel system consists of services built on the underlying microkernel. These services provide some functions that the kernel itself depends on, as well a basic set of user-level services for the construction of programs. The microkernel services can serve requests from multiple operating system personality clients and are used to construct the operating system personalities themselves.

In addition to the libraries that define the microkernel services, many libraries exist within the microkernel services that are part of the microkernel proper. These libraries represent the interfaces that the microkernel exports

and the support logic for the Message Interface Generator (MIG), which is used with the IBM microkernel's interprocess communication facilities.

A key element of the microkernel services environment is that it does not constitute a complete operating system. Instead, the microkernel services depend on the existence of a dominant personality, in this case OS/2 Warp Connect (PowerPC Edition).

The IBM microkernel is also dependent on some elements of microkernel services. There are cases in which it sends messages to personality-neutral servers to complete internal kernel operations. For example, in resolving a page-fault, the IBM microkernel may send a message to the default pager. The default pager then reads in the page that the kernel needs from a hard disk. Although the page fault is usually being resolved on behalf of a user task, the kernel is the sender of the message.

2.2.1 Initializing the Microkernel Services

The microkernel service for initialization consists of two distinct pieces:

1. An underlying Boot Loader (BL) that loads an image containing the microkernel
2. The bootstrap task

Boot Loader

The first program run when an IBM microkernel system starts is called the Boot Loader (BL). It is loaded by the firmware of the machine from the boot device into memory and then starts loading other programs and files into memory as proscribed by a configuration file that also resides on the disk. The configuration file contains the name of the microkernel to load, the name of the initial task, the names of other programs and files to load, along with any other information that is needed by later stages of the boot process.

In the IBM microkernel system, the initial task started by the boot loader is called the bootstrap task. This task is the first of the microkernel services tasks to be started.

Bootstrap Task

The bootstrap task has no device drivers built into it, and thus cannot access the hard disk. It can however, access information that the boot loader (BL) has already placed in memory. This information contains all that is needed for the bootstrap task to start tasks running.

Once it has started, the bootstrap task becomes a file server for the programs that it starts. As a file server, the bootstrap task serves out the other files that were read into memory by the Boot Loader.

The bootstrap task performs the following (in order):

1. Starts the Root Name Server.
2. Starts the Default Pager.
3. Starts the Task manager.
4. Provides File Services (which will be used by the Task Manager).
5. Directs the Task Manager to start to personality neutral (PN) servers required to bring up the dominant personality. PN servers include Message Logger, Hardware Resource Manager (HRM), Bus Walkers, and Device Drivers.
6. Starts the Personality.

The bootstrap task continues to behave as a file server until it terminates.

2.2.2 Task Manager

The task manager is a microkernel service that completes the boot process and can then be left to load programs or attach new libraries to already running programs.

The task manager maintains a set of event/action lists. The event/action lists describe what actions, usually loading a program, take place when a certain event occurs. The events usually contained in the event/action lists are name service notifications. The task manager requests that the name service notify it when certain changes are made to the name space. When the name service notifies the task manager that a change has occurred, the task manager scans through the event/action lists until a match is made against one of them. When the match is made, the indicated set of actions is taken.

This mechanism provides for an automatic configuration of the system based on what services are present in the system and what is needed by the system rather than by a fixed set of scripts. By providing such a mechanism, it becomes harder for the system to be configured incorrectly either by attempting to use services that are not present or providing services that will not be used.

There are five submodules in the task manager, as follows:

- **Extended Link Format (ELF) object loader**

  The ELF loader loads other microkernel services, such as microkernel services servers and device drivers.

- **Shared Library Management**

  Shared libraries can be linked at load time using the shared library management utility program.

- **Microkernel Services server management**

  The microkernel services server management module provides a set of APIs to load, start, and terminate microkernel services tasks.

- **Name Services**

  Name access service is provided to privileged system management software for retrieving task control ports, as well as other information.

- **Boot message logging**

  The boot message logging service provides a unified method for all microkernel services tasks to log their errors.

2.2.3 External Memory Managers

Memory objects are managed by memory managers. Unlike some virtual memory systems, memory managers are not part of the kernel, but are user mode tasks. Memory managers must register with the kernel. The memory manager and the kernel exchange send rights to two ports that are used by the kernel and the memory manager to communicate. The kernel calls the memory manager on one of these ports whereas the memory manager calls the kernel on the other port. However, creation, representation, and destruction of memory objects is a private responsibility of the memory manager. The kernel deals only with managing the cached pages of memory objects.

When a client wants to get access to a memory object or a portion of a memory object, this part of the memory object has to be mapped into the client′s virtual address space. The client does this by issuing a corresponding vm_map call. The kernel passes this request to the memory manager. The memory manager is notified by the kernel during the mapping process that the kernel is preparing to cache pages for a memory object. The memory manager declares the memory object attributes. Memory object attributes define the options and customized characteristics of the memory object.

When the system runs out of physical pages in order to satisfy a certain request, the kernel selects pages to be evicted from the cache to make room for memory object pages of greater demand. These pages are returned to the memory manager so that these pages can be removed from physical memory. Pages, that have not been updated can be discarded without being returned to the memory manager. Pages that have been updated are called ″dirty″ and are returned to the memory manager in order for the memory manager to update its abstract image of the memory object. Pages can be declared as ″precious″, in which case the pages are returned to the memory manager whether dirty or not.

2.2.4 Default Pager

The IBM microkernel provides a default memory manager, called the default pager, to manage temporary nonpersistent memory objects. The default pager has a paging space (backing storage) to hold the contents of these memory objects when the kernel needs to use physical storage for some other purpose. Memory backed by the default pager is called anonymous memory.

It is valid to have a memory manager that keeps its abstract memory object image in anonymous memory and does not stage the memory object to a backing store. In this case, the anonymous memory, which is managed by the default pager, may be evicted to the default pager′s backing space. This is called double paging and should be done, knowing that system resources will be used to maintain this data.

2.2.5 Root Name Server

The name space acts as a directory (or depository) to store and locate information about resources currently available in the system and which should be known to other components of the system.

The resources described in the name space include:

- Configuration information
- Port addresses
- Service providers
- Directories
- Files
- Personality Dependent (PD) resources
- Personality Neutral (PN) resources

Name Space

Definition:

- Permanent: Some parts of the name space are permanent from boot to boot unless the system configuration is modified.
- Transient: All other portions of the system are being defined as ″transient″ and are not saved and restored from boot to boot.
- Rules: With the exception of nodes attached directly to the root (ns_root_dir) of the global name space by the system at installation time, all name space nodes are transient unless they are specifically created as permanent.

Structure of the Name Space

The namespace of the IBM microkernel is a graph. Each graph is a collection of nodes. The nodes are connected by links. The entire name space exists as a directed graph. In many respects it also has the characteristics of a rooted tree. All name space nodes originate from a single root.

Most tasks make use of the namespace by doing one of the following:

- Find services or resources it needs.
- Advertise services or resources it manages.
- Being a name server itself.

The name space is not managed by a single server. As an example, there might be multiple file servers, each one managing a subset of the name space. The name server that manages the Root Directory is called the root name server. The collection of all name servers in a system implements the name space graph. These name servers might have different capabilities, that is, not all name servers implement the same set of application programming interfaces. Each name server owns a subset of a name space (sub-graph). Thus, the name space is ″fractured″, and these fractures are called name borders. Note, that there are no restrictions to force a tree structure, to preclude cycles, or to force the sub-graph to be connected.

Figure 3 on page 28 shows the structure of the root name space and its relationship to other private name spaces. Note, that the dotted line does not denote a link.

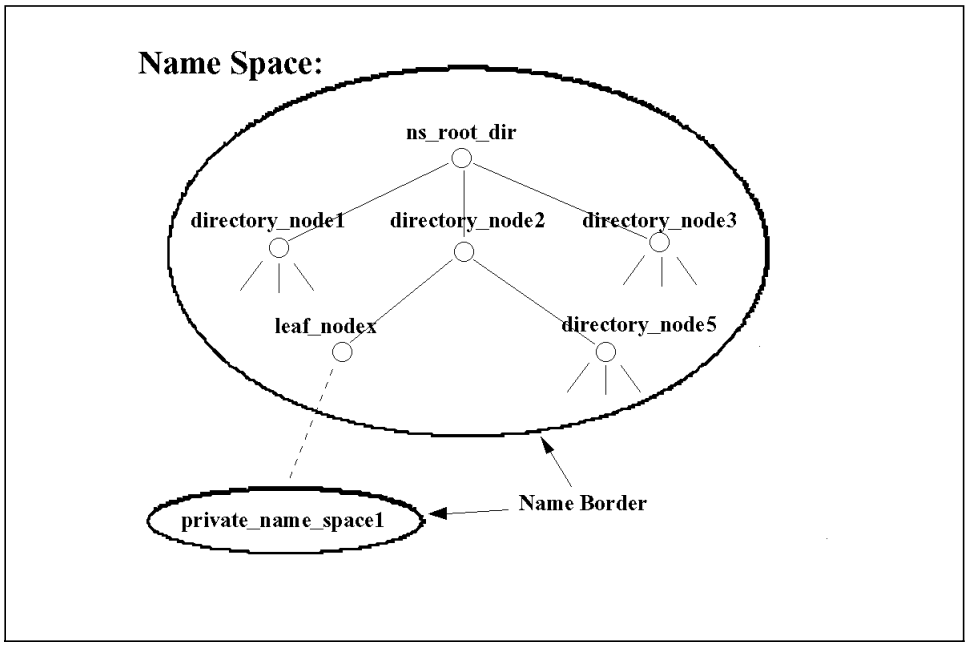


Figure 3. Root and Private Name Space

Links

A link maps a node and a name to a node. There are two restrictions placed on links:

- A link cannot cross name borders. The two nodes that are connected by a link must be managed by the same name server.
- The two nodes connected by a link must exist. If the node that the link points to is deleted, then the link is also deleted.

Nodes

Nodes are the data containers of the name space. Each node has a type, a set of zero or more attributes and an access control list (ACL). ACLs are the only mechanism used to protect the integrity of the name space, thus it is important to properly protect directory nodes. Attributes are used to store information and are pairs made up of a name and a value. The name is a possibly empty unicode string.

Thus, the name server provides more than just a naming service. It is also an information repository and a powerful query facility can be used to navigate the name space based on this information.

There are three types of nodes:

- **Directory nodes**

  Nodes that contain outgoing links to other nodes. However, directory nodes can be empty, that is, they have no outgoing links. Directory nodes can be used in much the same way that they are used in a file system. Related items are grouped together and organized within the name space. For example, different devices can be found under the devices directory. There is a special directory node called the root directory node (ns_root_dir) which has the following characteristics:

  – A handle to this node is inserted on all tasks.

  – It cannot be deleted.

  – It is created by the root name server at boot time.

  – Its ACLs are initialized in such a way that only trusted tasks can add or delete outgoing links.

- **Alias nodes**

  Alias nodes are used to reshape the name space. They provide a way to jump from one portion of the name space to another portion of the name space. To accomplish this, an alias node contains a reference to a node and a path. The node that the alias refers to must exist. When an alias is found during the name (for example path) resolution process, then the resolution process jumps to the node referenced by the alias and resolves the path stored in the alias before continuing with the resolution of the rest of the original path.

- **Leaf nodes**

  A leaf node does not have any outgoing links, that is it is the end of a name space path.

  A leaf node may or may not contain a send right, which can be given to a requesting task by the name server. If there is no send right, the leaf is just a placeholder for arbitrary information. If the leaf contains a send right, this points to a service provider associated with this leaf. The send right to this service provider allows the requesting task to communicate with the object or resource represented by this leaf. On the other hand, this service provider might also be another name server. This is the mechanism used to allow the name space to span multiple name servers (see the dotted line in Figure 3 on page 28).

The type of a node is determined at creation time and cannot be changed later on. Directories and aliases have the node type

NS_NODE_TYPE_DIRECTORY and NS_NODE_TYPE_ALIAS, respectively. Leafs, on the other hand, can have the generic type NS_NODE_TYPE_LEAF, or they can have a user-defined type.

Each node has a reference count. The reference count is incremented whenever a new reference to the node is established. When the reference count drops to zero, the node is deleted.

Anonymous Nodes

A node which has no incoming links is called an anonymous node, that is, it cannot be reached from the root directory. A task can create such an anonymous node in order to build a private name space, which is administered by the root name server. From such node an anonymous subgraph can be created, which is not part of the global name space and is therefore not known publicly. The anonymous directory can be shared by other tasks by having the task that created it provide a send right to the anonymous directory port to other tasks. Anonymous nodes and graphs can be used by peer processes as a means of interprocess communication.

Paths and Name Resolution

The concatenation of one or more link names is called a path. Paths are used to traverse the name space. In order to get a node handle, the requesting task has to provide:

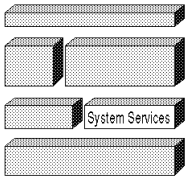
- A handle to a starting node
- A path

Given a starting node and a path, a simple, recursive algorithm returns the last node as a result of this name resolution algorithm.

Accessing the Name Space

Service providers need to modify the name space by means of an API in order to advertise services or to update system information. Therefore, to accomplish this task, a rich set of functions is provided by the root name server (for example, creating nodes/links as well as performing path resolution).

Chapter 3. System Services



The system services play a unique role in the OS/2 Warp Connect (PowerPC Edition) environment. They are not a part of the microkernel, nor are they part of the OS/2 Server. The system services are effectively neutral services that are utilized by the other components of the operating system. The advantage of this architecture means that the OS/2 server could be replaced, or additional servers could be added to the system, without having to recode the information that is part of the system services.

3.1 Device Support

One of the most important components of OS/2 Warp Connect (PowerPC Edition) is the device services. The device services, through the use of device drivers, provide access to the hardware for the other components of the OS/2 Warp Connect (PowerPC Edition) operating system.

To keep the microkernel small and to improve modularity and real time response, the device services can be implemented as user-level functions. This means that the device drivers, which in other operating systems are logically part of the operating system kernel, are application programs. They have privileges that other application programs do not have, such as having access to the physical device hardware, but they execute as standard tasks.

One of the advantages in having the device drivers in the user level of the operating system, is that it is possible to use conventional application development techniques to develop and test them. This aids in the reliability of the system since the failure of a single device driver does not imply a total failure, as is usually the case when device drivers reside in the kernel.

In the first release of OS/2 Warp Connect (PowerPC Edition), several of the system device drivers have been implemented to run wholly within the kernel. The reason for this has been to increase the performance of those device drivers over a user level implementation.

A limited set of device drivers have been supplied with the first release of OS/2 Warp Connect (PowerPC Edition). Support is provided only for devices originally supplied with IBM Power Personal Systems Series machines. The reason behind the limited availability of device drivers is that the current device driver architecture is due to be replaced by a proposed layered device driver architecture in a future release of the OS/2 Warp Connect (PowerPC Edition) system.

The device drivers supplied with this release of OS/2 Warp Connect (PowerPC Edition) include:

- Parallel port
- Serial port
- Diskette drive
- Console - This driver is actually a set of different drivers, including drivers for keyboard, mouse and the display adapter.
- Token-ring
- Ethernet
- PCMCIA Ethernet
- PCMCIA Token-ring
- SCSI
- IDE
- Audio

3.2 Event and Window Services

Event and Window Services (EWS) is the OS/2 Warp Connect (PowerPC Edition) mechanism for sharing the console device among applications. EWS handles screen groups, sessions and events.

A session is a collection of one or more tasks (or processes in OS/2). A session owns an input queue for keyboard and mouse input, and it owns a video buffer.

A session may have child sessions. Sessions maintain the state necessary to share console resources. A session may be active or inactive. Only active sessions can receive input events and be switched to the foreground.

A screen group is a session which controls the state of a physical video device, keyboard and mouse (or any other locator).

The events handled by EWS are essentially keyboard, mouse and pen events.

3.2.1 Screen Group and Session Management

The EWS is called to create and destroy sessions and screen groups.

The EWS maintains the session states and session interrelationships. The latter being parent/child, independent/dependent, bound/unbound, foreground/background and selectable/unselectable.

The EWS provides support for exclusive sessions such as Hard Error and Popups.

The EWS provides for switching screen groups, by notifying the screen groups being switched in an out of the foreground.

The EWS notifies screen group owners, session owners and session watchers of session management events. The EWS allows programs to register as session watchers. As such, they will be notified about session management events. An example of a session watcher, is the OS/2 tasklist.

3.2.1.1 Session Manager

The Session Manager component is responsible for:

1. Maintaining the list of active sessions
2. Maintaining the z order of screen groups
   The z-order determines the layering of windows on the screen.
3. Notifying session watchers (tasklist) about session changes (creation, deletion, switch and title change)
4. Providing the session management API

The session manager receives all its requests as microkernel interprocess communication messages generated from the session management API of OS/2 Warp Connect (PowerPC Edition). The session manager deals with three types of requests:

1. Popup display requests
2. Hard error display requests
3. All other requests

The session manager maintains internal queues to hold the requests. Popup and Hard error requests are served before the session switching requests.

3.2.1.2 Sessions

A session has a session owner, it has a video resource and it has an input queue for receiving keyboard and locator (mouse) input.

Session owners in OS/2 Warp Connect (PowerPC Edition) are the OS/2 Server and the Multiple Virtual Machines component. A session owner receives a notification when a session terminates. The session owner is responsible for maintaining a list of the processes belonging to the session, and for terminating them.

Sessions may be related to other sessions in parent/child relationships. Children may be bound to their parents. Selecting a session with a bound child, results in the child session being brought to the foreground instead of the parent.

3.2.1.3 Shutdown Services

On a normal shutdown, EWS is called twice by the OS/2 Server. The first time it is called, EWS is requested to send a message to all screen groups and sessions, that the system is about to shutdown. This allows the involved session owners and applications to terminate normally. They may also cancel the shutdown. If none of the session owners request cancellation of the shutdown, the OS/2 Server will call event and window services a second time, and this time EWS will notify the screen groups and sessions that the shutdown is imminent.

3.2.2 Event Services

The primary responsibility of the event services component of event and window services is to facilitate high-level sharing of console input. It runs as a thread of the event and window services task, reading messages posted by the micokernel to the event and window services input port. The messages received, are basically keyboard and mouse messages. However, other sources may also send messages to the input port. An example could be a pen server sending pen-created input formatted as keyboard messages, so the event services component would not know the true source of the messages.

The primary job of the event services component is to translate the input events (keyboard, mouse, pen, etc.) into a common event packet, maintain shift state and pointer position, and route the message to the current input queue.

In OS/2 Warp (Intel), this was done in the keyboard and mouse device drivers and in the PMWIN.DLL. The event services component of event and window services provides a common location to place all input handling.

3.2.2.1 Input Port Messages

Five types of messages are allowed on the input port of the event services. These messages can come from the console device driver, from another device driver simulating a console device driver, or from a program simulating keyboard or mouse input.

The five message types allowed, are:

1. Keyboard scancode
   This is the basic keyboard event as received from the console device driver. It consists of a packet containing a timestamp and a single byte of type 1 (AT enhanced) scancode.
2. Locator record
   This is the basic mouse event as received from the console device driver. It consists of a packet containing a timestamp, mouse button action, and position information.
3. Control
   The control event message is used by device drivers to send status change requests to event services. A control event message indicates a state change in the physical device, and requires that event services update its local state.
4. Multi-event
   The multi-event message is designed to allow a programmable interface to all of the above mentioned events. In addition, the multi-event allows input of unicode characters, virtual keys and PM scan codes.
5. Notification
   Notification events are sent to event services from other threads within event and window services. These are not meaningful if sent from any other place. Notifications such as session termination are sent this way.

Logical Devices:

Input events can come from either real devices or simulated devices, such as a pen device, or the special needs component of event and window services (see 3.2.2.4, “Keyboard Special Needs” on page 39). When the data comes from a simulated device, it is necessary to understand how the state of the real device affects the simulated events. Event services define the logical devices as a means of specifying these relationships. Each session has

three logical console devices defined. These can be used to maintain separate settings, such as shift, for the various logical devices. This is mostly useful for a program wishing to simulate keyboard and mouse events without changing the state of the real shift and button states.

The logical devices are:

- Device 0
  This defines the real device. Any changes made to this device are affected by the state of the real device.
- Device 1
  This defines a long term logical device. Users of this device should be careful to complete a set of actions. For instance, any keys which are pressed should be released.
- Device 2
  This defines a transient logical device. Users of this device should include a EV_RESET control at the start of each record, which sets the state to match the physical device and a known shift state.

Multi Event Input:

Events from real keyboard and locator devices tend to consist of a single action. Simulated events are often more complicated, and consist of a series of events which must be synchronized.

For instance the simulated event may consist of a mouse move, a button down and a series of keystrokes.

Multi-events consist of a header followed by a series of two byte values which are interpreted sequentially. Some control values (such as the scan code input) represent a single event. Other control values (such as unicode character) contain a length where many following values are data values.

The following types of events can be sent using the multi-event interface:

- **Unicode Characters**
  Unicode characters are sent with a control value which gives the count of characters, followed by a string of unicode characters. The characters are translated to complete input event packets and sent to the application as if they where entered from the keyboard.
- **Codepage Characters**
  Characters in the current keyboard codepage can be sent with a control value containing a count, followed by a string of characters. Each character is contained in a two byte field. The characters are translated

to complete input event packets and sent to the application as if they where entered from the keyboard.

- **Virtual keys and Deadkeys**
  Virtual keys and Deadkeys are sent using the two byte VK_ or DK_ value as a single event. This sends a make/break of the key and has no permanent effect on the shift state. The resulting virtual or deadkey is translated and sent to the application as if entered from the keyboard.

- **PM Scancodes**
  PM translated scan codes are sent by OR-ing the scancode with the desired make/break code. It is sent as a single event value. Four make/break codes are defined:
  EV_SCAN, EV_SCANDOWN, EV_SCANUP and EV_REPEAT.
  The scancode is sent through keyboard translation and to the application as if entered from the keyboard.

- **Type 1 Scancodes**
  These are the native scancodes of the PC/AT keyboard, with additional support for the enhanced keyboard. This is the scancode sent by the keyboard. It consists of a single byte, where the highorder bit is the break indicator. A scancode of 0XE0 indicates that the next byte is an extended scancode.
  The scancode is sent through keyboard translation and to the application as if entered from the keyboard.

- **Locator Buttons**
  The buttons are normally associated with the mouse, although devices like pen, tablet and trackball are also supported. Button events are sent as a single event value, indicating the make/break status and the number of the relevant button. Event services support 32 buttons.
  The event is sent to the application as a mouse event.

- **Locator Position**
  position events are sent as a control value followed by a set of dimensions. A mouse would have two dimensions, but other devices might have more. The position can be either relative or absolute. The absolute dimension must be in the coordinate space of the locator and is translated to video coordinates by the event services.
  Several event types are defined:
  EV_RELMOVE, EV_ABSMOVE, EV_RELPOS and EV_ABSPOS. These all require the number of dimensions to be specified. Two dimension version of the above mentioned event types are also defined. This is for example EV_RELMOVEXY.

These events are translated and sent to the application as mouse events. If the mouse is currently being drawn on the display, it is redrawn in the current position.

- **Control Events**
  Control events come with or without data values. Control events without data consist of a single control value. Control events with data consist of a control value containing a length, followed by some data values.

3.2.2.2 Keyboard Translation

Keyboard translation is done using a set of global scancode translation tables and by calling the Universal Language Support (USL) keyboard functions. Scancodes arrive as either type 1 scancodes or PM scancodes. Type 1 scancodes are translated to PM scancodes with a simple translation table. This logic also detects repeat keys. The result of this step is a PM scancode with an indicator of make/break/repeat.

The ULS keyboard function is called to update the shift state, and the resulting scancode and shift state are used to generate the BIOS scancode. If the keyboard LED state is changed, the keyboard device driver is called to do the update.

The ULS keyboard function is called to generate the unicode character and the virtual key. The unicode character is used to construct the codepage character.

When the input is a character, the ULS keyboard untranslate function is used to translate the character to a scancode.

Keyboard layouts are defined by the ULS component, and are global to all session owners (OS/2 Server and Multiple Virtual Machines Server) in OS/2 Warp Connect (PowerPC Edition). They may be changed at any time, and applications may simultaneously be using different keyboard layouts. These keyboard layouts are user definable and created using the keyboard compiler (makekb).

Hotkey Processing:

After a key is translated, and before it is routed to the current input queue, a check is made to see if it is a hotkey. This can be done only after shift translation is done. Then keys are matched against the hotkey table within event and window services. If the key is a hotkey, the associated port is notified.

Hotkeys may be global (for instance CTL-ALT-DELETE) or session (for instance CTL-BREAK). Event and window services support an API call to set hotkeys.

3.2.2.3 Locator Conversion And Pointer Painting

The coordinates of the locator must be converted to absolute coordinates in the coordinate space of the current video mode.

The locator coordinate mapping logic can be replaced with a user function. This function is an entrypoint in a shared library, and is called with the current locator position, coordinate mapping information and the locator event.

In OS/2 Warp (Intel), the mouse pointer is painted as part of the mouse interrupt. In OS/2 Warp Connect (PowerPC Edition), the locator pointer is painted when the event is processed by event services. The actual painting is done by sending a message to the session owner, and the session owner's paint function (for instance PM if the session owner is the OS/2 Server) will do the actual painting.

A paint request is only sent when it is necessary. The decision of when to paint is based on the time since the last painting and distance moved.

Event and window services allow each session to specify an interest rectangle. This rectangle is set up by the session owner, and is used to allow suppression of certain locator events, either inside or outside of the rectangle.

3.2.2.4 Keyboard Special Needs

Some people have difficulties using the traditional keyboard and mouse so that they need special accomodations. The event and window services keyboard special needs component provides these accomodations.

This support is modeled closely on the industry standard AcessDOS from the Trace Center at the University of Wisconsin. The functional names used in this section are the names used in AccessDOS.

The following functions are supported:

- **StickyKeys**
  Allows the user to press each key of a multiple key operation separately.
- **RepeatKeys**
  Sets the keyboard repeat rate to a slow rate, or turns it off all together.

- **SlowKeys**
  Sets the sensitivity of the keyboard by not accepting a key until it is held down for a certain period of time.
- **BounceKeys**
  Prevents a key which is quickly repressed from being seen as a double press on the key.
- **ToggleKeys**
  Use a sound to indicate that a toggle key has been pressed.
- **MouseKeys**
  Use the keys on the numeric keypad to simulate a mouse.
- **SerialKeys**
  Use an external device attached to the serial port (or other character device) to act as an additional keyboard.

3.3 File Services

The file services provide file system support to the rest of the OS/2 Warp Connect (PowerPC Edition) operating system. The file services are based on a *file services framework*, which provides the model on which the components of the operating system can access the file services.

The File Services framework consists of both File Services Client interface and the File Services Server. A client is an application running as a user task together with the File Services-related libraries linked to the application. The OS/2 Server is an example of a File Services Client.

The File Services Server task includes the Logical File System, the File Server Pager, and a Physical File System and runs in user space. In this context, the term client/server does not mean that a machine-to-machine relationship exists, instead it is derived from the message passing architecture of the MK. Any application task that interfaces with the File Services Server is considered to be a File Services Client.

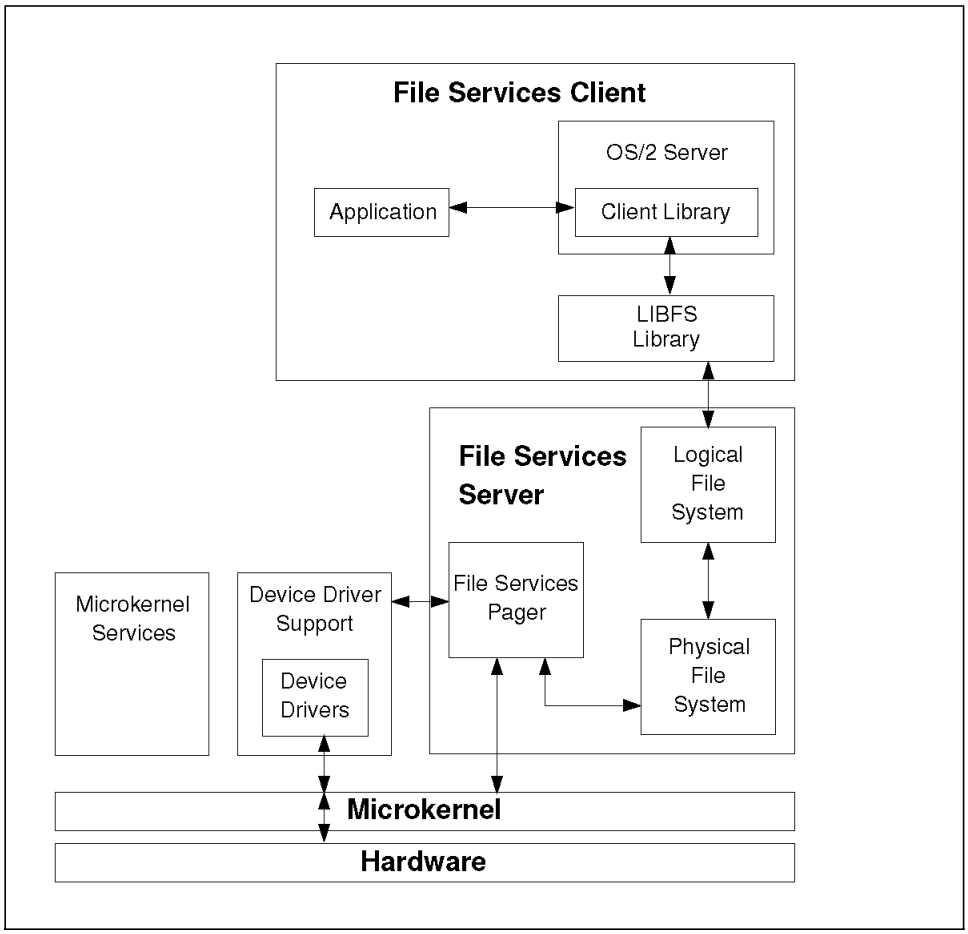


Figure 4. File Services Framework Overview

3.3.1 File Service Client

The File Services Client interfaces part of the File Services framework is the task which originates the file system request. An application submits a file system request (using native OS/2 calls), which gets translated into a file system server request. The LIBFS library, which is part of the File Services framework, then sends this request to the File Services Server.

3.3.2 File Services Server

The File Services Server part of the File Services framework includes the Logical File System, the File Services Pager and the Physical File System. This framework is designed to allow many different Physical File Systems (from within IBM or from other vendors) to be plugged in with minimal effort. In the first release FAT, HPFS and a CD-ROM Physical File System are supported. IBM plans to encourage other vendors to port their Physical File Systems to this framework.

3.3.2.1 Logical File System

The Logical File System provides path resolution, notifications, attributes, tokens support and common resource management.

- Path resolution resolves a path to the correct File Services entity (directory or file).
- Notifications allow a client to request a message when a particular type of change is made to the userdata or metadata maintained by the File Services.
- Attributes are additionally named and typed data to be attached to a File Services entity.
- Tokens allow client applications to explicitly map files and control access while mapping.

3.3.2.2 File Services Pager

The File Services Pager handles all paging activity on behalf of its File Services Server. This includes receiving page requests and page returns from the IBM microkernel, and handling all I/O to the device drivers. The File Services Pager provides a buffer manager abstraction to the Physical File System that is much simpler than the IBM microkernel′s external memory manager interface (EMMI). Because the File Services Pager does all I/O for the File Services Server, it also shields the Physical File System from knowledge of the device interface, and the thread and port interfaces. Using the FS pager allows for a global shared cache that is much more dynamic than traditional file system caching mechanisms.

3.3.2.3 Physical File System

A Physical File System manages the on-disk storage, indexing, mounting and recovery. A Physical File System is a part of the File Services framework. Instead, a Physical File System is the service provided to the File Services framework.

A Physical File System may exploit the IBM microkernel global cache mechanism, if so the Physical File System must also use the File Services Pager′s interface for buffer management instead of using its own private pinned buffer cache. The Physical File System has the option of performing the device I/O. This is intended for those special cases where the device does not fit in the conventional storage system architecture.

The only differences between different Physical File Systems that can be seen by an application are performance, degree of data integrity, and any function not supported by a particular Physical File System.

3.3.3 Thread and Port Model

The components of the File Services are executed on a thread and port model that is provided by the File Services framework. Ports are used in the communication between the File Services Server, the File Services Client, the device drivers and the microkernel. Threads are used to multi-thread the activity of the File Services Server.

3.3.4 File Services Pager

The File Services Pager is one of the external pagers for OS/2 Warp Connect (PowerPC Edition). It is the only component in the File Services Server that is communicating with the hardware through the Device Services The File Services Pager consists of components to control the usage of memory, to handle page-in and page-out requests and to interface with the other parts of the File Services Server

The File Services Pager and its related components have the following key responsibilities:

- Handling of all paging requests returns from the microkernel for the memory objects that it owns.
- Handling of all input/output required to process a page-in or page-out request for the memory objects that it owns.
- Support mapping of memory objects into the address space of the File Services Server and the address space of the File Services Client.

3.3.5 Physical File System (PFS)

The File Services Framework is designed to allow many different Physical File Systems to be plugged in with minimal effort. The Physical File System is a service provider to the File Services Framework.

Each Physical File System must include a set of Physical File System utilities to support CHKDSK, FORMAT, RECOVER and SYS.

If the Physical File System contains files needed to boot the system, it will need to have a Boot-PFS included.

Physical File Systems included in OS/2 Warp Connect (PowerPC Edition) are: FAT, HPFS and CD-ROM.

3.3.5.1 Physical File System Interfaces

The Physical File System attaches to the framework as part of the File Services Server task. The Physical File System must interface with the following three major system components:

- **The Logical File System part of the File Services Server**

  The Logical File System uses the Virtual File System++ interface to communicate with the Physical File System. The original Virtual File System interface was introduced by Sun Microsystems to make it possible to implement different file systems within one UNIX kernel. This interface, or variants of it, appear in most UNIX kernels today. The Open Software Foundation (OSF) made some enhancements to this interface to support their DCE Distributed File System. The OSF interface is called Virtual File System+. The Virtual File System+ interface is defined in an OSF document titled ″DCE DFS VFS+ Interface Specification″ from Transarc Corp.. Virtual File System++ (VFS++) is an extension of the Virtual File System+ (VFS+) interface. Currently, these extensions are primarily associated with additional operations to support Unicode and OS/2 Warp Connect (PowerPC Edition) user-defined attributes.

  The Physical File System uses the vnode pool that is provided by the Logical File System and jointly maintained by the Logical File System and the Physical File System.

- **The File Services Pager or external memory manager**

  The File Services Pager handles all paging and input/output for the Physical File System, shielding the Physical File System from the more complex interfaces of the microkernel and the device drivers and also provides caching. The microkernel includes threads, ports, and memory-object management. The Physical File System has the option of

making use of the name cache facility, also provided in the Physical File System library.

- **The microkernel interfaces**

  The interfaces to the microkernel are used to allocate memory objects.

  All length and offset parameters are 32 bit in this release of the OS/2 Warp (PowerPC Edition). Until the microkernel provides a 64 bit memory object, the File Services will use 32 bit memory objects. This places a restriction on OS/2 Warp Connect (PowerPC Edition) allowing the maximum file size of 32 bits. When the microkernel provides a 64 bit memory object, the File Services will then be changed to use the 64 bit memory object, increasing the maximum file size to 64 bits.

3.3.5.2 File System Utilities

The utilities described here are those that require knowledge of the Physical File System layout.

- **CHKDSK**

  CHKDSK analyzes a file system or volume for layout integrity and produces a disk status report.

- **FORMAT**

  FORMAT prepares the specified media with a file system layout. After FORMAT has completed, the specified file system may use the media for normal read/write activity.

- **RECOVER**

  RECOVER recovers portions of a file from a disk that has defective sectors. RECOVER works with single and multiple files from the same volume.

- **SYS**

  SYS adds files to a specified volume to enable booting.

- **DISKCOMP**

  DISKCOMP compares the contents of the diskette in the source drive to the contents of the diskette in the target drive.

- **DISKCOPY**

  DISKCOPY copies the contents of the diskette in the source drive to the diskette in the target drive. If necessary, the target drive diskette is formatted during the copy.

In addition to the listed functions, FDISK also runs from the Utility File Services Framework. DISKCOMP and DISKCOPY only support FAT. FDISK does not require any knowledge of the Physical File System layout and therefore only one FDISK is required. All other utilities are part of the OS/2 personality.

Utility File Services (UFS):

To support the CHKDSK, FORMAT, RECOVER and SYS utilities OS/2 Warp Connect (PowerPC Edition) provides the Utility File Services. The Utility File Services is part of File Services and functions in a similar manner as the File Services The OS/2 personality is responsible for the front-end function to each of the utilities supported by the Utility File Services. In addition the front-end calls the specific utility entry in the Utility File Services which sends the necessary parameters. The Utility File Services provides a set of entry points that the client uses to access the functionality of the utilities.

The Utility File Services are divided into three sections: the Utility File Services client library (libufs), the Logical Utility File Services (LUFS), and the Physical Utility File Systems. The libufs exports the UFS_ APIs to the client. The LUFS contain a routing mechanism used to dispatch the utility call to the proper Physical File System implementation. The Physical Utility File Services contain the worker routines which understand the Physical File System′s on-disk structure. The Physical Utility File Services communicate with the Device Drivers and the microkernel to perform its duties.

During the execution of these utilities it is necessary for the Physical Utility File Services and the front-end utility code to communicate with each other. This communication is to provide status and obtain additional user input.

To support this communication, the front-end utility code will establish a port which will be used to receive messages. This port is passed to the Physical Utility File Services as one of the API parameters. The Physical Utility File Services will respond to the front-end utility code passing its message port. A two-way communication is now established.

3.3.5.3 FAT, HPFS and CD-ROM Physical File Systems

The implementation of the FAT and the HPFS Physical File System is based on the standard DOS and/or OS/2 file system layout including OS/2 extended attributes. The design takes into account compatibility with standard FAT and OS/2 functionality.

The CD-ROM Physical File System is the link between the data on the CD-ROM drive itself and the File Services Logical File System layer. It

supports regular (ISO 9660) CD-ROMs at this time. OS/2 Warp Connect (PowerPC Edition) also supports booting from CD-ROM on hardware where the BIOS supports this.

3.3.6 Volume Manager

The intent of the Volume Manager is to encapsulate the details associated with the identification and accessing of the different storage volumes available to the system. It simplifies the requirements on the File Services Server by isolating it from variations in partition schemes and by providing a repository for pertinent partition/volume information.

It additionally can serve as a centralized component for general volume management function, ranging from the mundane task of obtaining the data associated with the separate volumes to more sophisticated volume management features such as volume spanning, striping, mirroring, etc.

The first implementation of the Logical Volume Manager is called Basic Volume Manager (BVM) and is implemented to meet the initial requirements, with consideration given to future expansion.

The functions that are provided by the Basic Volume Manager are the following:

- Upon invocation at IPL, the Basic Volume Manager spawns the volmgr server and inserts the appropriate node under the servers branch of the root name server. Subsequent invocations will fail when the presence of the volmgr node is detected.
- The Basic Volume Manager creates and maintains the volumes subtree under the root name server
- The Basic Volume Manager creates a logical device instance for all valid and recognized partitions (volumes).
- The Basic Volume Manager provides an interface that allows external parties to query and/or notify the Basic Volume Manager of changes in the status of any volume subtree entries, and it updates those entries appropriately.
- The Basic Volume Manager monitors the root name server devices subtree and the logical device instances created for the tracked volumes and performs the appropriate updates on the root name server volumes subtree as devices/partitions/volumes are changed.

3.4 Pipe Services

The pipe server is a personality neutral server that provides OS/2 Warp Connect (PowerPC Edition) applications with the abstractions required for interprocess communication through the use of pipes. The IBM microkernel′s message passing architecture through one way message passing and Remote Procedure Call (RPC), provides the basis for the pipe server, with some additional capabilities provided to the pipe server clients, for example, named communications channels, reading the pipe data in a user specified format, queueing up for a pipe till it becomes available for use, remote pipe support, among others.

The purpose of the pipe server is to provide a basic set of interfaces for creating and using pipes, without attempting to be specific to any particular operating system′s implementation of pipes. The API set is therefore generic, and any further specific functionality required by OS/2 Warp Connect (PowerPC Edition) will have to be implemented in an emulation library. The OS/2 Warp Connect (PowerPC Edition) emulation library provides the OS/2 pipe APIs with specific functionality such as:

- Peeking into a pipe to look at the data in it, without removing the data from it.
- Associating a semaphore with a pipe, to synchronize read and write operations on a pipe.

Pipe Server and the Name Space

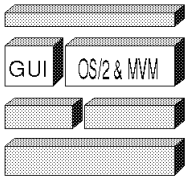
Upon initialization, the pipe server will register itself with the root name server which will provide a service port, that can be used by any client that needs to communicate with the pipe server.

When an application wants to create a pipe, the pipe server registers this pipe in the root name server under the pipe server′s directory on behalf of the requesting application. Thus, the name space tree regarding pipes is created and controlled by the pipe server. The name space for the pipes consists of the pipe tree which has an entry for each named pipe in the system.

As pipes can also be used as a means of communications between remote machines, the pipe server will also register remote pipe names under the name space for the corresponding redirectors (like IBM LAN, Novell). This allows the pipe server to determine which redirector to request services from, for an opening of a specific pipe on a remote server. In order to keep

an updated list of network redirectors, the pipe server requests the root name server for notifications about changes to the tree of redirectors in the name space.

Chapter 4. OS/2 Functions



This chapter describes the basic OS/2 functions known from OS/2 Warp (Intel). They are 4.1, “OS/2 Server,” 4.2, “The MVM Environment” on page 72, 4.3, “Graphics Subsystem” on page 87, and 4.5, “Printing Services” on page 119. There is also a section describing the systems management functions of OS/2 Warp Connect (PowerPC Edition), 4.6, “System Management.” on page 122.

4.1 OS/2 Server

The OS2 Server is designed to provide the OS/2 Warp 3.0 API (the API calls with prefix DOS) on behalf of OS/2 Warp Connect (PowerPC Edition).

It is assumed that the reader is familiar with the basic functionality of the OS/2 Warp (Intel) kernel, also called the OS/2 Control Program. For more information about the OS/2 Control Program, see *OS/2 Version 2.0. Volume 1: Control Program*, GG24-3730 .

4.1.1 OS/2 Server Architecture

The OS/2 Server architecture is based on a client/server model that makes use of microkernel interprocess communication (IPC). The client side of the model is OS/2 Warp 3.0 applications. The applications communicate through the API to a set of DLLs, which are responsible for providing the APIs, either directly or by requesting service from the server side of the model. Figure 5 on page 52 depicts the base API calls implementation.

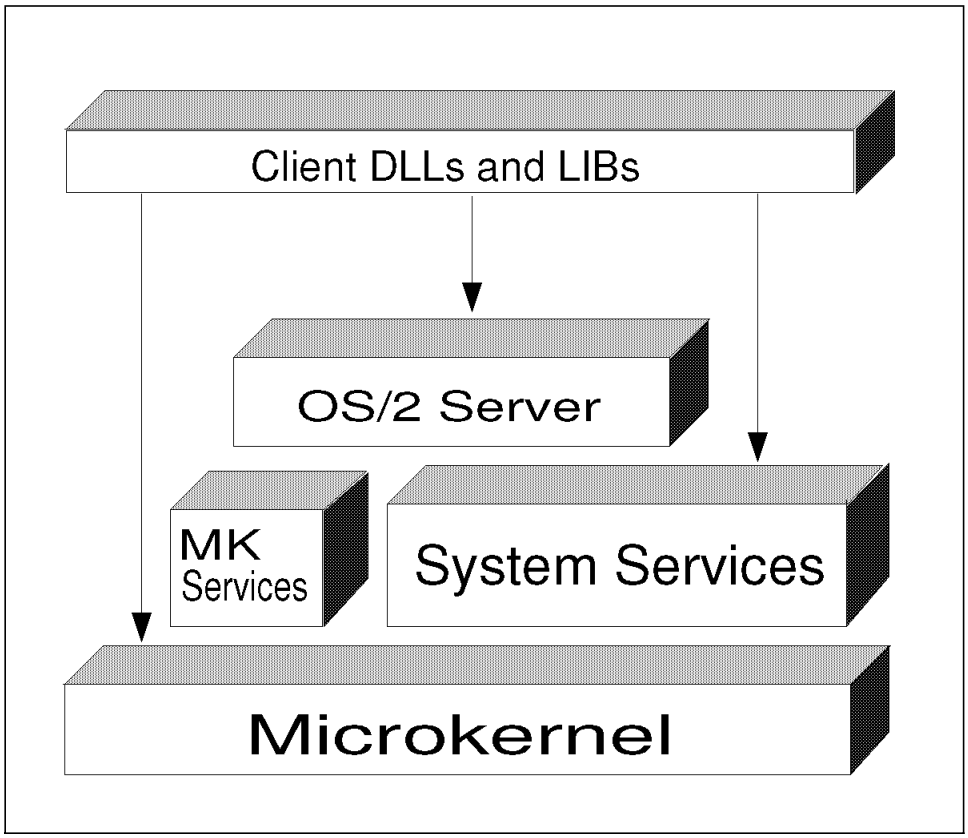


Figure 5. Base API Calls Implementation

4.1.1.1 Client Side

This section looks at the general aspects relating to the client side of the client/server model mentioned earlier. When we describe the different components of the OS/2 Server in more detail, we see that there might be client side issues to be discussed.

The client side is a collection of DLLs and libraries, bound to OS/2 user processes. The DLLs are as follows:

- **DOSCALLS.DLL**

  This DLL contains the most commonly used API calls with prefix DOS.

- **QUECALLS.DLL**

  The entry points for the Queue related API calls are found in this DLL.

- **LIBMK.DLL**

  This DLL contains library routines for the microkernel system calls.

- **LIBCXPG.DLL and LIBCMXPG.DLL**

  These libraries contain the common ANSI C runtime routines. The system calls in LIBMK.DLL are used to implement some of the functions in LIBCXPG.DLL and LIBCMXPG.DLL.

- **FSCALLS.DLL**

  This library contains the API calls to the file services in OS/2 Warp Connect (PowerPC Edition), implemented as a shared service.

The base API calls to the OS/2 Server are resolved by the loader into entrypoints in the DLLs on the client side.

Some of the API calls are handled entirely within the client side DLL, while other calls result in messages being sent to the microkernel. The majority of calls are made as remote procedure calls to components of the OS/2 Server.

4.1.1.2 Server Side

This section looks at the general aspects relating to the server side of the client/server model mentioned earlier.

The OS/2 Server consists of a single multithreaded task (task is the microkernel term equivalent of an OS/2 process). For performance reasons, the OS/2 Server is multithreaded.

The following are the main threads:

- One initial thread

  Which performs the initialization and spawns other threads.

- Multiple message threads

  These threads read client requests, in the form of IPC messages from a microkernel port set. When finished with a request, a reply is sent back to the request originator.

- One exception thread

  Which reads exceptions raised by the microkernel.

- Multiple pager threads

  Which handle page faults.

- One semaphore timeout thread

Which implements semaphore timeout functionality.

- One timer timeout thread

  Which implements timer timeout functionality.

- One control port read message thread

  Used to spawn other tasks from external servers.

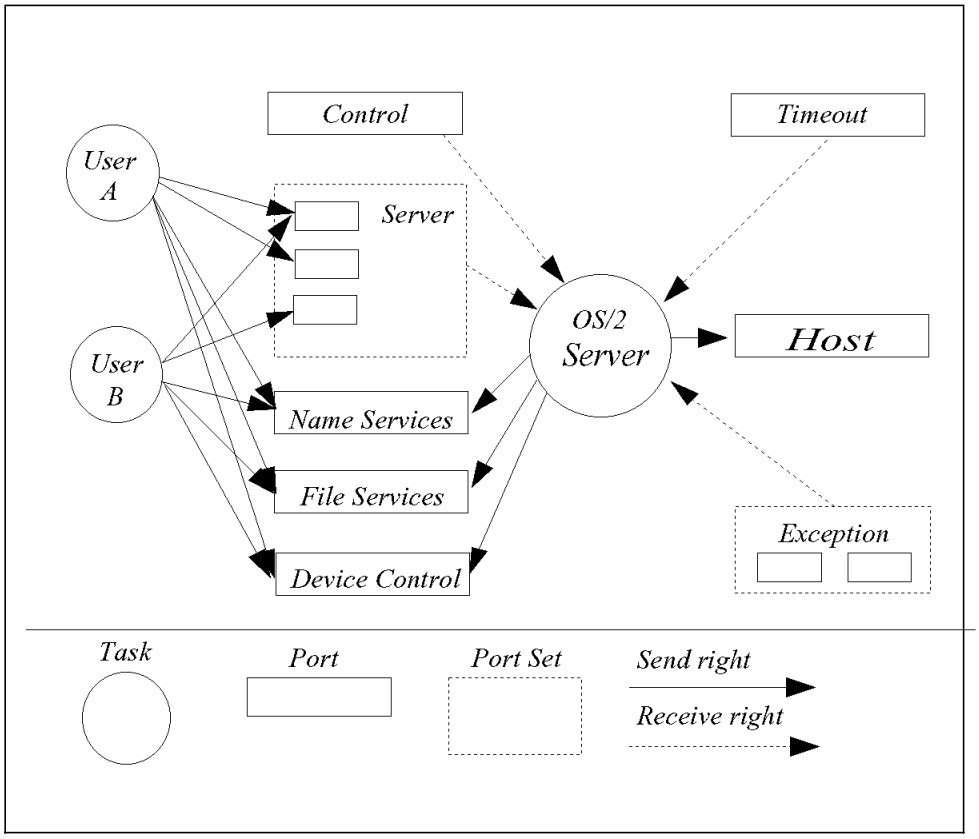


Figure 6. Main Interfaces of the OS/2 Server

Figure 6 shows the microkernel ports used by the OS/2 Server in order to interface with other components of OS/2 Warp Connect (PowerPC Edition).

2.1.3, “Inter Process Communication (IPC)” on page 10 describes the interprocess communication supported by the microkernel and used by the components of OS/2 Warp Connect (PowerPC Edition). Whenever a task (or OS/2 process) or thread is created, it is associated with a microkernel port. These ports are not depicted in the figure.

- **Control Port**

  The OS/2 Server receives messages on this port from other servers in the system wanting to run OS/2 programs.

- **Server Port Set**

  The OS/2 Server has read rights to this port set, and uses it to read requests from user processes.

- **External Server Ports**

  Used by the OS/2 Server to send messages to external servers, such as name services, or File Services.

- **Device Control Port**

  Privileged microkernel port used to access devices.

- **Timeout Ports**

  The OS/2 Server receives messages on these ports from the microkernel, when requested timers are expired.

- **Host Ports**

  These ports are used to inquire and set the system clock, and to alter scheduling policies on behalf of threads.

- **Exception Port Set**

  A port set where the OS/2 Server receives exception messages from the microkernel on behalf of any user thread.

4.1.2 Configuration

The OS/2 Warp Connect (PowerPC Edition) environment defines a system name space, which contains persistent object information for all the OS/2 Warp Connect (PowerPC Edition) environment including configuration information for the OS/2 Server. In OS/2 Warp (Intel), configuration information is found in the CONFIG.SYS file. No API is present in OS/2 Warp (Intel), with the purpose of updating the CONFIG.SYS file. All updates are made directly to the CONFIG.SYS file. This is common practice, when installing application programs. It makes the boot process of OS/2 Warp (Intel) vulnerable, because a successful boot is dependent on the contents of CONFIG.SYS.

The CONFIG.SYS file still needs to be maintained in OS/2 Warp Connect (PowerPC Edition) for compatibility with applications that use the file directly. OS/2 Warp Connect (PowerPC Edition) however, only uses the following statements during the boot process: SET, RUN and RUNSERVER.

The SET and RUN statements, are the ones we know from OS/2 Warp (Intel), while RUNSERVER is new in OS/2 Warp Connect (PowerPC Edition). RUNSERVER starts only privileged services. All RUN statements will take place after all the RUNSERVER statements.

The RUNSERVER command takes the following parameters:

```
RUNSERVER=<program>
                          {-arg "arguments"}
                          {-lookfor <name space entry>}
                          {-timeout <seconds>}
```

The RUNSERVER statement synchronizes the startup of a server program based on the parameters provided. By default the RUNSERVER will wait until the program terminates. If the option -lookfor is specified with a name, the RUNSERVER will wait until the name has been created in the system name space. If some maximum time has been specified with the -timeout option, the RUNSERVER will wait this maximum amount of time before returning. If both -lookfor and -timeout are given, the RUNSERVER will return as soon as one of the options is satisfied, or a completion notification is received (the program has terminated).

The following RUNSERVER statement shows how the event and window services may be started:

```
RUNSERVER=C:\os2\ews.exe -LOOKFOR servers\SessionClient -TIMEOUT 20
```

4.1.3 Components Of The OS/2 Server

This section looks at the different components of the OS/2 Server. We start with the description of handle management, because it is common to many of the components we describe later in the section.

4.1.3.1 Handle Management

The concept of a handle is well known from OS/2 Warp (Intel). After an initial API call, giving the name of a requested resource, when successful, OS/2 returns an entity known as a handle. All subsequent API calls referring to the mentioned resource, must use the handle to identify the resource to OS/2.

Handle management must manage two types of handles. The handles on the client side in DOSCALLS.LIB and the handles on the OS/2 Server side.

Figure 7 on page 57 shows two user applications (represented by PTDA 1 and PTDA 2) and their involvement with the handle management.

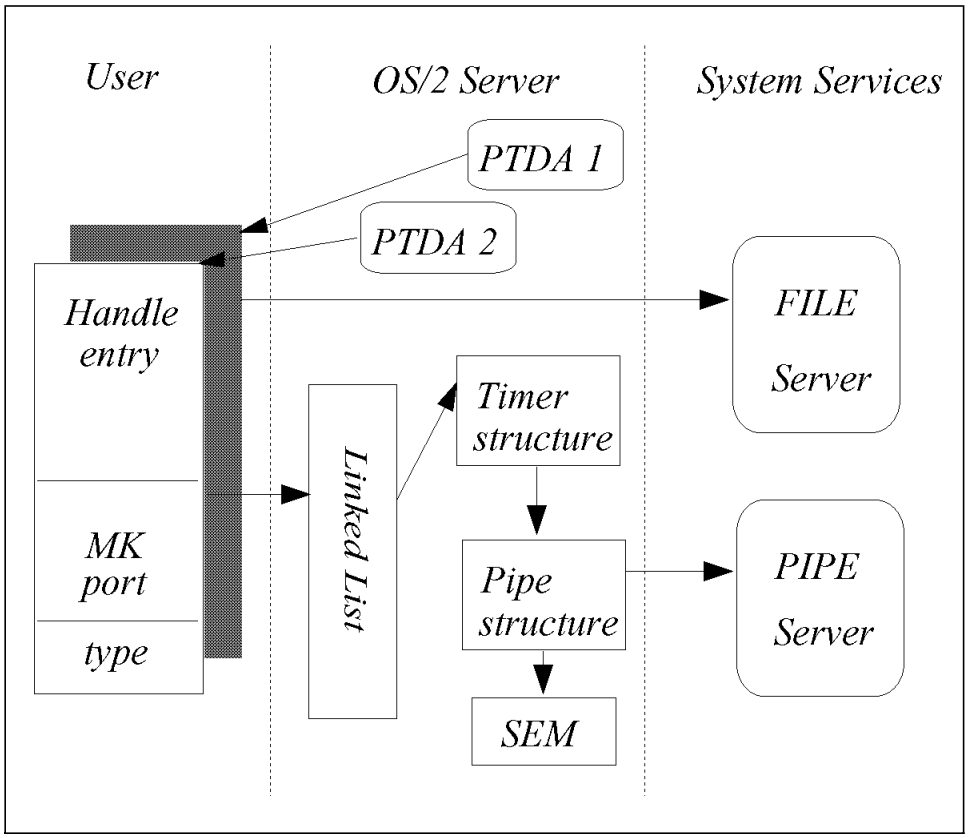


Figure 7. Handle Management Examples

- Client side handle management

  The handle management on the client side maps the OS/2 handle to a microkernel port and a handle type. The OS/2 handle is actually an index into a table. There is one handle table per process, and handles can be inherited by child processes. The handle type is used to identify the component serving the object referenced by the handle. This component will either be routines residing entirely on the client side, or it will be a component of the OS/2 Server or it may be a system service.

  In Figure 7, we see that the client part of the application requesting file services (represented by PTDA 1), communicates directly with File Services. The client side of the other application (represented by PTDA 2), points to the handle table on the server side, and eventually winds up communicating with the pipe server. Notice also that the linked list based on the microkernel port, also contains a timer data structure, indicating that application 2 also has a pending timer request.

The semaphore and the queue component of the OS/2 Server have their own handle management.

- Server side handle management

  The handles on the OS/2 Server side are microkernel ports. There is one global handle table on the server side, which is an array of pointers. The microkernel ports are hashed to get their position in the table. Each pointer points to a linked list of handle table entries, with the last pointer being null. Each handle entry is part of the data structure associated with the component being managed, so the address of a component′s data structure can be found by getting the address of its table entry via the microkernel port.

The handles managed by the handle management component, and how they participate on the client and/or server side are:

- File - only client side interaction
- Device - only client side interaction
- Pipe - both client and server interaction
- Timer - only server side interaction
- Process - only server side interaction
- Thread - only server side interaction

4.1.3.2 Session Support

Refer to 3.2, “Event and Window Services” on page 32, to get a definition of a session.

The session component of the OS/2 Server provides the API calls which implement the functionality of the OS/2 Warp (Intel) session management. In OS/2 Warp Connect (PowerPC Edition), session management is the responsibility of the event and window services. When the OS/2 Server receives a session request, it will work in conjunction with the event and window services to fulfill the request. Event and window services will be started by the OS/2 Server during startup.

4.1.3.3 Tasking

The tasking component of the OS/2 Server is responsible for the management of OS/2 processes and threads. This include process and thread creation, control and termination. The tasking component is also responsible for providing process and thread information and for handling thread synchronization through thread suspension or allowing threads to wait on other threads. Finally, the tasking component is responsible for

implementing the process and thread API as we know it from OS/2 Warp (Intel). The tasking component interacts with other OS/2 Server components including memory management, the loader, and the semaphore system. Its main interface is the microkernel, because it directly uses the microkernel task and thread support, see 2.1.4, “Tasks and Threads” on page 16.

The tasking component of the OS/2 Server has the following key responsibilities:

- Process creation and thread creation
- Maintain information about process and thread specific data
- Control and provide information about current processes and threads
- Process and thread termination and cleanup

Process and Thread Creation:

An OS/2 process maps directly to a microkernel task. A process is started by a DosExecPgm call to the tasking component. A control block called the Per Task Data Area (PTDA) is created on behalf of the process. An initial thread is created on the behalf of the process.

An OS/2 thread maps directly to a microkernel thread. A thread is created by the DosCreateThread call to the tasking component. A thread information block (TIB) structure is initialized and associated with the thread.

Process and Thread Information Maintenance:

Each process has a PTDA associated with it. The PTDA is kept in the address space of the OS/2 Server.

The main content of the PTDA is the OS/2 handle of the process, the associated microkernel port and the address of the TIB for the initial thread.

As each new thread is created, a TIB is associated with it. The TIB is kept in the address space of the OS/2 Server and it contains port information, priority information, thread stack information, and the rest of the machine state information.

Process and Thread Query and Control:

The DosGetInfoBlocks API call provides information to an application about its process and its current thread. The DosSetPriority call enables applications to alter the priorities of their threads.

The tasking component supports synchronization between a process and its child processes. It supports synchronization between threads, so that threads may wait on other threads, threads may suspend other threads and threads may kill other threads. Also the DosEnterCritSec/DosExitCritSec calls are supported.

Process and Thread Termination and Cleanup:

Process and thread termination is managed by the tasking component of the OS/2 Server. When the initial thread of a process is terminated, the process is terminated.

When a thread terminates, its TIB is released. When a process terminates, its PTDA is released. A process may have registered exit processing. In this case the tasking component is responsible for managing the exit list processing.

4.1.3.4 Memory Management

The memory management component of the OS/2 Server has the following key responsibilities:

- Manage private/shared memory areas for OS/2 applications
- Manage page faults for guard pages and executable objects
- Manage the growth and shrinkage of the paging space used by the microkernel′s default pager.
- Implement memory related OS/2 API

The memory management component of the OS/2 Server uses the microkernel virtual memory management functions to implement the OS/2 memory management semantics. OS/2 applications expect certain behavior with regard to memory allocation and shared memory, such as all shared memory must be loaded at the same virtual address in all participating processes.

The memory manager component takes advantage of the microkernel′s External Memory Management (EMM) support to free it from page management, and instead let it focus on memory object management.

See 2.2.3, “External Memory Managers” on page 25 for more information on the EMM. The EMM will direct a page fault to the appropriate page fault handler for that particular page. The page fault handler would be the default pager, the OS/2 loader or the File Services, depending on the context of the faulted page.

Private and Shared Memory for OS/2 Applications:

The memory manager component uses arenas to manage private and shared memory for OS/2 processes. An arena is a circular-linked list that is used to keep track of memory allocation in the address space of an OS/2 process. The information contained in the arenas is just what is needed to extend the microkernel memory semantics to OS/2. The virtual address space supported by OS/2 is a 4GB linear address space with its layout (see Figure 8).

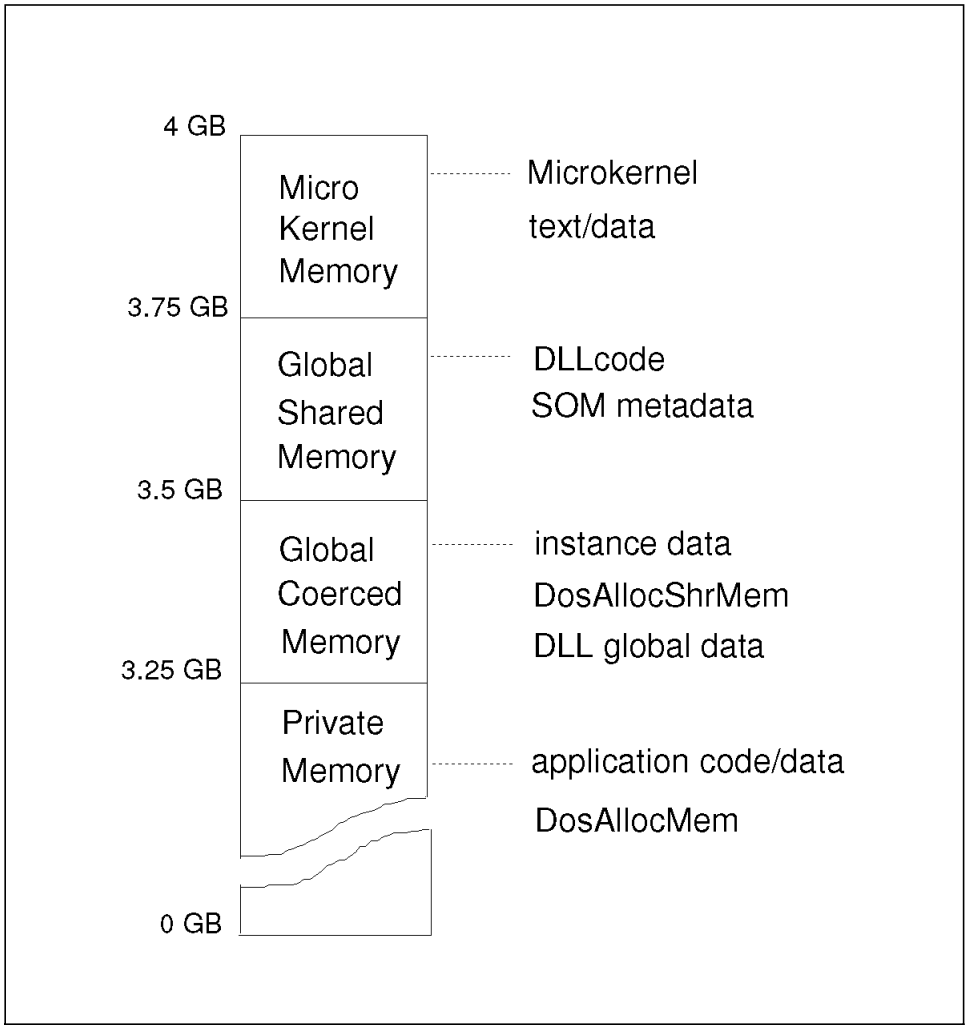


Figure 8. Virtual Address Space Layout

An OS/2 process may request private memory either by using the DosAllocMem call or by using the C runtime support. The C runtime support of malloc, realloc, etc. is not using the memory management component, but calls the appropriate microkernel functions directly.

The Global Shared Memory (GSM) contains memory that is immediately accessible to all processes in the system. The user processes will only have read/execute access to the GSM, while servers (for example the loader) may have write access.

The Global Coerced Memory (GCM) is used where there is need for more restricted access to memory objects. In the GCM area, memory is allocated in blocks, where each block may have different access attributes for each process with addressability to the block.

Page Faults for Guard Pages and Executable Objects:

The manager of page faults on behalf of the memory manager is called the OS/2 pager. The OS/2 pager is responsible for managing page faults for pages that contain compressed data (for example Presentation Manager resources) and stack guard page faults.

Compressed data page faults are handled by invoking the loader to load the appropriate information from the executable file and decompressing it.

Stack guard page faults are handled by simply allocating more stack space via the exception handling component, see 4.1.3.10, “Exception Handling” on page 64.

All other page faults are handled by the default pager or File Services.

Growth and Shrinkage of Paging Space:

The growth and shrinkage of the default pager’s paging space is the responsibility of the OS/2 Server. It does so by issuing appropriate API calls (DPAGER_XX...) to the default pager.
The OS/2 Server must:

- Monitor the utilization of the paging space via DPAGER_Set_Threshold. The default pager will send a notification message to the OS/2 Server when the threshold is reached.
- Allocate additional space to the paging space when the utilization of the paging space has reached 80%. This is done by the call DPAGER_Add_Space.

- Remove excess space from the paging space when the utilization of the paging space drops below 50%. This is done by a call to DPAGER_Remove_Space.
- Log errors to the system log

Implement Memory Related OS/2 API:

The memory manager component of the OS/2 Server handles all the memory related API calls, with the exception of the Memory Suballocation Package (MSP). The MSP is implemented solely in the DOSCALLS.DLL client library.

The memory manager component uses the microkernel virtual memory manager functions in order to implement the memory related API. The calls for memory allocation need to be identified to the microkernel. This is done by sending the task port of the OS/2 process doing the API call, with the call to the microkernel.

The process of identifying the originator of the API call and manipulating the message parameters is the job of the Message Interface Generator code.

4.1.3.5 Semaphores

The semaphore component of OS/2 Warp Connect (PowerPC Edition) is ported directly from OS/2 Warp (Intel). The only alteration necessary, is to the DOSCALLS.DLL, which now uses microkernel interprocess communication to communicate with the semaphore component of the OS/2 Server.

4.1.3.6 File I/O Support

The OS2 Server is not involved with the file I/O requests made by the applications. The file I/O API calls in DOSCALLS.DLL are converted to the FS_XX... API calls of File Services.

4.1.3.7 Pipes

The main effort regarding pipe creation and maintenance is with the Pipe Server running as a system service. An application calls a pipe API residing in the DOSCALLS.DLL. Parameters are validated within the DLL. If OK, the request is shipped to the pipe server and handled there. Upon completion of the request, a process handle on the client side is generated or updated (if it already exists). See 4.1.3.1, “Handle Management” on page 56 for an explanation of handle management.

If the API request involves a semaphore, or the pipe content is of type message, the OS/2 Server participates in the pipe management process, otherwise the pipe is managed solely by the pipe server.

4.1.3.8 Queues

The queues component of OS/2 Warp Connect (PowerPC Edition) is just ported from OS/2 Warp (Intel), with the necessary alterations to the QUECALLS.DLL to use microkernel interprocess communication to the queue component of the OS/2 Server.

4.1.3.9 Timer Management

The timer management component of the OS/2 Server is responsible for implementing the API calls DosSleep, DosGetTime, DosSetTime and the start and stop timer API calls.

The DosSleep call and the DosGetTime call are implemented by directly calling the similar function in the microkernel from the DOSCALLS.DLL.

The DosSetTime function is handled differently, because it must perform a privileged operation on the microkernel. In this case the OS/2 Server is called via the RPC interface, and the OS/2 Server does the necessary call to the microkernel, through the Host port in Figure 6 on page 54.

The timer calls are handled as in OS/2 Warp (Intel), apart from the fact that the timeout notifications are handled by the microkernel by sending messages to the timeout port of the OS/2 Server. The notifications are sent as a result of preceding set alarm calls made by the OS/2 Server to the microkernel.

4.1.3.10 Exception Handling

The exception handling component of the OS/2 Server is responsible for emulating all the exception related API calls and also for handling exceptions raised by the microkernel. Exceptions that arrive at the OS/2 Server can have two distinct origins:

- Microkernel Raised Exceptions
  These are the actual microkernel exceptions that the OS/2 Server receives by a thread of the OS/2 Server called the system exception server.

  Examples of these exceptions are all the hardware generated exceptions, such as memory access violations, divide by zero, etc. The OS/2 system exception server receives these exceptions, translates them into OS/2 style exceptions and calls the appropriate routines to handle them. One

special exception handled by the system exception server, is the guard page fault exception (see “Page Faults for Guard Pages and Executable Objects” on page 62).

The system exception server simply allocates more stack memory on behalf of the failing thread.

- Client Side Raised Exceptions
  These exceptions are not exceptions from the microkernel point of view, but are actually OS/2 exceptions resulting from DOSXXX API calls on the client side. These API call generated exceptions are handle by a component of the OS/2 Server called the application exception server.

4.1.3.11 Message Management

The message management component of the OS/2 Server enables application programs access to messages kept in separate files outside of the applications. This component is ported directly from OS/2 Warp (Intel) The utility MKMSGF is ported and also the linker must support the binding of the message support segment created by the MKMSGF utility.

4.1.3.12 Debugging Support

The debug component of the OS/2 Server supports two interfaces to its system services. These interfaces are the DosDebug API known from OS/2 Warp (Intel), and the new Debug Probe. Debug support enabled by the debug probe is the capability to follow the flow of an application request across all server calls and into the microkernel.

The debug probe provides an interface to the OS/2 Server for debuggers which require information specific to the OS/2 Server. The primary interface is a set of request/reply messages required to debug or trace debuggee programs.

The API calls available to debuggers are similar to the DosDebug API calls.

4.1.4 Loader

All tasks (or OS/2 processes) in the system after the initial tasks loaded by the bootstrap loader (see 4.1.5.1, “Bootloader” on page 67) are loaded by a single loader, the OS/2 server loader, in cooperation with the OS/2 Server memory management component.

The OS/2 Warp (Intel) semantics of global shared memory (text and data) at the same virtual address in every task’s address spaces is preserved. This is true for all tasks including virtual DOS machines, device drivers and system services.

Some of the design goals for the OS/2 loader are:

- Use a single library binary and a single virtual copy of libraries loaded by both OS/2 processes and system services.
- Have a single loader in the system at any point in time. Preferably, this loader should have been implemented as a system service.
- Provide global shared memory services for those tasks (and OS/2 processes) that require them.

The loader is responsible for the loading of code and data from executable modules on disk on behalf of an OS/2 application. The loader supports 32-bit little endian modules in the Executable and Linking Format (ELF) format. See *AT&T UNIX System V Release 4 Programmer's Guide: ANSI C and Programming Support Tools* for more information about ELF.

The responsibilities of the loader can be summarized as follows:

- Loading of OS/2 executable and OS/2 dynamic link libraries and system service libraries at OS/2 process creation time at the request of the tasking component of the OS/2 Server.
- Unloading the OS/2 executable and OS/2 dynamic link libraries and system service libraries at OS/2 process termination time at the request of the tasking component of the OS/2 Server.
- Loading and unloading of OS/2 dynamic link libraries and system service libraries as requested by the OS/2 application.
- Cooperating with the tasking component of the OS/2 Server to supply per process and per system library initialization and termination.
- Cooperating with the debug component of the OS/2 Server to support the DosDebug API.
- Providing access to resources as requested by the OS/2 application.
- Providing the API for loading, unloading and querying dynamic link libraries.
- Allowing OS/2 applications access to functionality exported by system services.

4.1.5 Startup

This section describes the OS/2 Warp Connect (PowerPC Edition) boot sequence resulting in the microkernel and the OS/2 Server running. The OS/2 Server subsequently starts up the rest of the OS/2 services including the user interface and the Multiple Virtual Machines. Also the event and window services is started by the OS/2 Server.

4.1.5.1 Bootloader

The boot process begins when the bootloader image is brought into memory by the PowerPC firmware.

The image in memory contains the bootloader program, the boot device driver and the file system extensions needed to access the boot media. The device driver and the file system support in the bootloader can change depending on the installation (for example the boot device is a CDROM device and the installation target is a hard disk). The device extensions and file system extensions are dynamically linked to the bootloader to allow it to read files for each boot device.

There is a distinction between the boot device and its file system extension and the paging device and its file system extension. They may be the same for a particular boot of the system, but if the boot device is a CDROM device, the paging device would be assigned to a different device.

The OS/2 Warp Connect (PowerPC Edition) installation for release 1.0 requires two disk partitions, a small hidden partition required by the PowerPC firmware, and the system partition. Chapter 5, “Installation” on page 129 will provide you with more information about installation requirements.

The hidden partition only contains the bootloader, which is loaded and started by the firmware.

The system partition contains the microkernel binaries, the base paging space, the Basic Volume Manager, the registry, all of the OS/2 services and configuration files such as BOOT.CFG and CONFIG.SYS, as well as stanza files describing the hardware.

A paging file managed by the OS/2 server will also be on the system drive. The bootloader is responsible for loading programs and files into memory and creating and maintaining a data structure to be delivered to the microkernel when it is started by the bootloader.

The bootloader reads the BOOT.CFG file from the system partition. BOOT.CFG is an ASCII file containing information about programs and files to be loaded into memory by the bootloader.

BOOT.CFG may contain entries PN_BOOT_DEV and PN_BOOT_FS that specify the storage device and the file system to be used for reading the remainder of the files loaded by the bootloader. The entries may appear multiple times, allowing the boot information to be collected from different places.

A PN_BOOT_CONFIG statement in BOOT.CFG gives the name of a configuration file that contains additional configuration information.

The bootloader has a fail safe boot recovery capability in the case of bad boot information. The bootloader will use the last good copy of BOOT.CFG in this case.

After the bootloader has finished its task (loading programs and files into memory), it starts the microkernel. The microkernel starts the bootstrap task.

4.1.5.2 The Bootstrap Task

The bootstrap task has all of the files loaded by the bootloader in its virtual address space. The bootstrap task knows which tasks to start via the information it obtains from the microkernel through the host_get_boot_info interface. The microkernel, on the other, hand got the information directly as a data structure from the bootloader.

The bootstrap task starts the tasks as identified by the PN_FILE_NAME entries in BOOT.CFG. The bootstrap task knows if a file named in a PN_FILE_NAME is a program to be started, because a program must have at least one PN_FILE_ARG=<string> entry. If no argument is required, the <string> is just a null string.

The order of the startup of the individual components is important. They are started in the order they appear in the BOOT.CFG file, due to dependencies on previously started components. The bootstrap task waits for a started component to register itself with the root name server and to place its service port within the root name server.

The root name server itself is a special case to the bootstrap task, in that the bootstrap task waits for a message from the root name server (after having started it), containing the root name server service port. The bootstrap task

provides the root name server service port to all services is starts. The bootstrap task also registers its file services port with the root name server.

Services Started by the Bootstrap Task:

As already mentioned, the root name server is started and its service port kept by the bootstrap task. The root name server enables name services to be present for all components started subsequently. The name space entries which need to be persistent, will not be made persistent until the registry server is started later in the boot sequence.

The default pager is started with no paging space available. It registers itself with the root name server. The base paging space is allocated by the dominant personality pager initialization task (PAGRINIT), after the paging device driver and File Services are started later in the boot sequence.

In release 1.0, Device Configuration Services consists of the Hardware Resource Manager (HRM), Configuration Manager and Device Manager. HRM is started first, and it uses stanza files to recognize the hardware in the system. Stanza files are ASCII text files containing descriptions of the various hardware components. Next, the Configuration Manager is started, followed by the Device Services. Device Services calls the API LDR_XX... to start the device drivers. The drivers are then placed in memory by the bootloader.

The bootstrap task now starts the OS/2 initialization, the Basic Volume Manager and the File Services. It is now ready to start PAGRINIT, because the file system is available. After paging is enabled, the registry is started. Finally, the bootstrap task creates the OS/2 Server task and starts the OS/2 Server.

4.1.6 Shutdown

The intent of a shutdown is to quiesce applications and system services in preparation for rebooting or powering off the system. Therefore, programs receiving a shutdown message, should make sure that any volatile data should be saved before it terminates. A program does not need to terminate, and could in fact request that shutdown be halted. An OS/2 Server thread controls the shutdown process.

4.1.6.1 Shutdown Invoked by User or API

The user or an OS/2 application can request a shutdown of the system. This is done via a Workplace Shell desktop selection, or via the APIs WinShutdownSystem and DosShutdown. WinShutdownSystem is an orderly shutdown for OS/2 applications and system services that elect to be notified of a pending shutdown. DosShutdown simply flushes the file system cache and perform other requested actions such as system dump and reboot.

WinShutdownSystem:

The Workplace Shell gets the desktop shutdown request from the user and issues a WinShutdownSystem call to Presentation Manager. A Presentation Manager application may also call WinShutdownSystem. Then the Presentation Manager calls the OS/2 Server.

The OS/2 shutdown thread:

- Calls event and window services to shutdown screen groups and sessions as described in 3.2, “Event and Window Services” on page 32. A nonzero return code makes the shutdown halt.
- Sends a shutdown message to each server that registers its service port in the \server or \device name space nodes with an attribute of ShutdownMessage=Yes. Servers or device drivers that do not have this attribute will not be informed about the shutdown.
  - The shutdown thread sends shutdown messages to each server followed by shutdown messages to each device driver. An attribute of ShutdownOrder={range 0-255} can be specified to control the order of notification. Servers with a value of 0 are notified first, while servers with a value of 255 are notified last. Servers with the same value have no implied ordering.
  - The File Services shutdown, ShutdownOrder=128, will flush any disk caches and disable further write operations, but the File Services will remain operational and able to handle page faults.
  - These shutdown messages are done as remote procedure calls. The shutdown thread will wait for each server to respond. In release 1.0, the number of servers requesting notification are small, and they are trusted to respond.
- Calls event and window services again to indicate that shutdown is complete or that it is cancelled.

- Returns control to the caller of WinShutdownSystem (the Workplace Shell or the Presentation Manager application) which displays a message to the user that it is safe to power off or reboot the system.

DosShutdown:

The DosShutdown API has been extended to allow a Software Distribution Manager like NetView/DM to initiate a file system shutdown, followed by a reboot. System dump and halt options have also been added.

DosShutdown can perform the following actions:

- Send a Shutdown message to the File Services to flush buffers to disk and quiesce the File Services. The File Services will disable further write operations but will remain operational and able to handle page faults.
- Perform a system dump, reboot the system or halt the system as required.
- Control is returned to the caller which can then display a message informing the user that it is safe to turn off the power or reboot the system. If a dump, halt or reboot is requested, control is not returned if the request is successful.

In order for a system dump to be taken, the location of the dump file must be specified to the microkernel by a systems management utility.

The dump, halt and reboot actions are initiated through the host_reboot call to the microkernel, using the Host Control port, with RB_DUMP, RB_HALT or RB_REBOOT option specified.

4.1.6.2 Shutdown via CTRL-ALT-DEL

The shutdown thread of the OS/2 Server performs the following actions when it receives a CTRL-ALT-DEL notification:

1. Sends a message to the screen, indicating rebooting.
2. Sends a Shutdown message to the File Services to flush buffers to disk and quiesce the File Services. The File Services will disable further write operations but will remain operational and able to handle page faults.
3. Reboots the system via the host_reboot call to the microkernel, using the Host Control port.

4.1.6.3 Abnormal Termination of a Shared Service

In release 1.0, there is no provision for restarting a failing system service The architecture allows for a monitoring of servers through the port_death notification, so that the OS/2 Server might restart failing system services.

4.2 The MVM Environment

OS/2 Warp Connect (PowerPC Edition) includes binary support for DOS, Windows or DPMI applications. In this, the first release, the operating system provides DOS support that is based on the OS/2 Warp (Intel) version of the product.

The Multiple Virtual Machine (MVM) environment provides the following functionality to the OS/2 Warp Connect (PowerPC Edition) operation system:

- Boots a DOS emulation kernel in a Virtual DOS Machine (VDM).
- Boots a non-emulated version of DOS (for example DRDOS, Novell DOS, DOS V3.3 and above) in a Virtual Machine. In OS/2 Warp (Intel), this is referred to as a Virtual Machine Boot (VMB) .
- Runs multiple DOS applications concurrently in the system.
- Runs DOS applications fullscreen or windowed.
- Provides DOS, Windows, DPMI applications access to a common set of file systems shared with other personalities in the OS/2 Warp Connect (PowerPC Edition) environment.
- Provides virtual device support to allow sharing of devices between DOS, Windows, DPMI applications and OS/2 applications.
- Provides LIM EMS 4.0 support.
- Provides LIMA XMS 2.0 support.
- Provides DPMI 0.95 host support in the system.
- Provides support for Windows 3.1 applications by running WIN-OS/2 from OS/2 Warp (Intel) in a virtual machine.
- Provides support for network redirection of drives in a VDM.
- Provides support for user configuration of the MVM environment on a per VDM basis.
- Support Win32s applications as found in OS/2 Warp (Intel).

The MVM environment provides the above listed functionalities in the OS/2 Warp Connect (PowerPC Edition) system with the help of several components

from the Service Layer of the system. The service layer is a term to describe services or facilities available in the operating system, but which are not part of either the OS/2 Server or the MVM environment.

The components that the MVM environment uses are:

- **OS/2 Loader**.

  The OS/2 loader is used for loading programs of attaching libraries to already running programs.

- **HRM**.

  The Hardware Resource Manager portion (HRM) is used to request and get hardware resources for a virtual machine.

- **File Server**.

  The file server provides file system services to Virtual Machines.

- **Event and Session Manager**.

  For session management and keyboard and mouse events.

4.2.1 OS/2 Warp (Intel) Multiple Virtual DOS Machine

The Multiple Virtual DOS Machine (MVDM) architecture in OS/2 Warp (Intel) is designed to exploit the virtual 8086 (V86) mode of the 80X86 processor, which allows operating systems such as OS/2 Warp (Intel), to execute multiple DOS applications within the 80X86 protected mode environment.

The MVDM Kernel controls the state and the architecture of concurrent VDMs, and is composed of the following four major components as shown in Figure 9 on page 74.

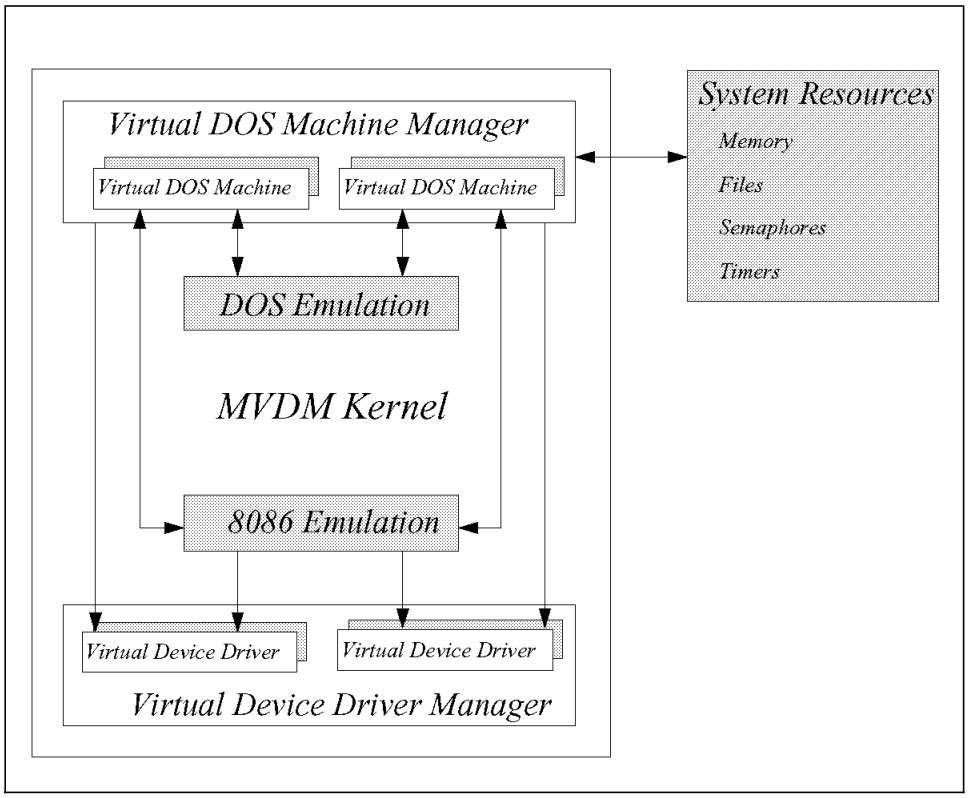


Figure 9. MVDM Architecture

1. **The Virtual DOS Machine Manager (VDMM)**

   This component contains the mechanism to start and interact with DOS applications. It creates, initializes, and terminates Virtual DOS Machines (VDM).

2. **8086 Emulation**

   The 8086 emulation manages communication between 8086 instruction streams and virtual device drivers.

3. **DOS Emulation**

   This component emulates the function and operation of the DOS operating system on a per-VDM basis. DOS services are emulated with the MVDM kernel, or by invoking protected mode services provided by the OS/2 kernel.

4. **The Virtual Device Driver Manager (VDDM).**

   The VDDM loads, initializes and communicates with virtual device drivers. Virtual device drivers are required to virtualize the hardware and ROM BIOS, thereby allowing DOS applications to access hardware device and BIOS without affecting other applications in the system.

4.2.2 OS/2 Warp Connect (PowerPC Edition) MVM Environment

The MVM environment is the facility that provides DOS, Windows or DPMI support in the OS/2 Warp Connect (PowerPC Edition) operating system. Based on the OS/2 Warp (Intel) Multiple Virtual DOS Machine (MVDM) architecture, the MVM environment is built from many of the same components as the MVDM. Figure 10 shows the MVM environment architecture.

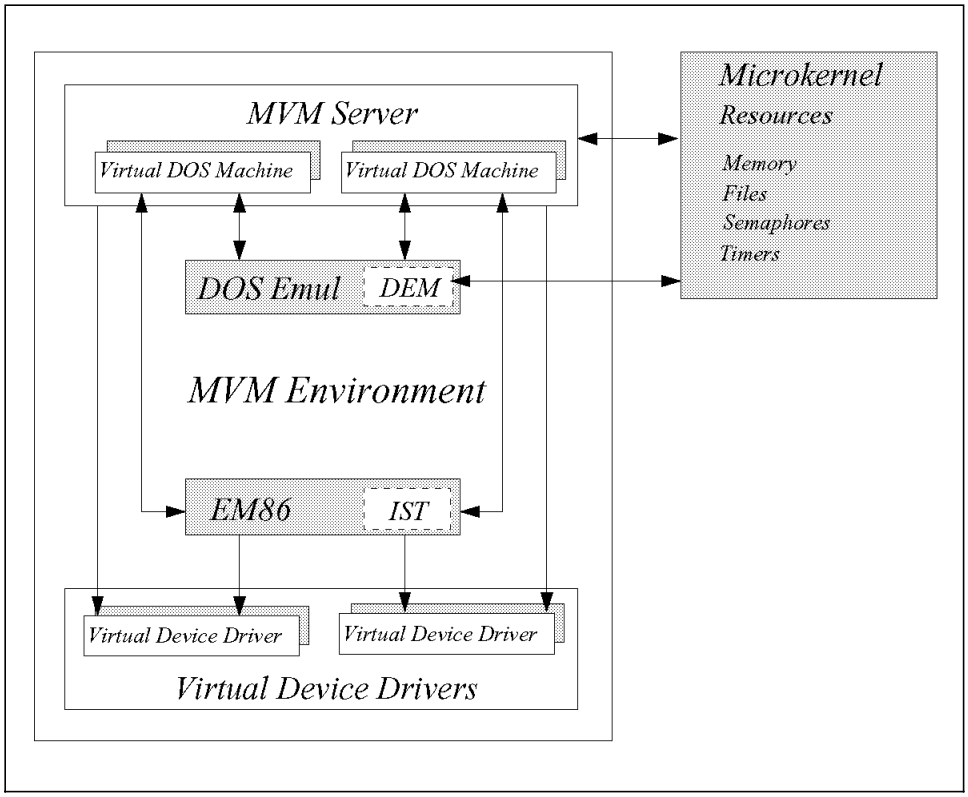


Figure 10. MVM Architecture

Obviously, with the change to the PowerPC platform, the MVM environment had to grow in order to support the new requirements of the PowerPC

processor. For example, without the Intel 80X86 processor to cater for Intel instructions, OS/2 Warp Connect (PowerPC Edition) provides an Instruction Set Translator (IST) to do the job.

The MVM environment is comprised of the following four major components:

- **MVM Server**

  A Multiple Virtual Machine server that exports a message based API to create, configure and destroy VDMs. The server maintains a database of global and per virtual machine configuration and tuning properties.

- **8086 Emulation or EM86**

  The EM86 component services all exceptions in a VDM, decodes the instruction that caused the exception and dispatches an appropriate Virtual Device Driver to emulate or virtualize it.

- **DOS Emulation**

  This component emulates the function and operation of the DOS operating system on a per-VDM basis. This component utilizes a DOS Emulation Service Interface (DEM) in order to access microkernel resources.

- **Virtual Device Drivers**

  A collection of Virtual Device Drivers (VDDs) that virtualize or emulate different aspects of the PC and X86 environment for the VDM.

4.2.3 Installation

The MVM Environment can be installed during the Personality Feature Installation Phase of the OS/2 Warp Connect (PowerPC Edition) install process. This means that the MVM can be installed during the OS/2 Warp Connect (PowerPC Edition) installation, or later as part of an additional installation of features. More details about the OS/2 Warp Connect (PowerPC Edition) installation is available in section 5.2, “Feature Install” on page 131.

For the purpose of installation, the MVM environment is composed of three main components. They are:

- Multiple Virtual DOS Machine (MVM) server, which supports the Virtual DOS Machines (VDM)
- WIN-OS/2
- Win32s

Although you can install the MVM environment using the defaults, you can customize which of the sub-features of the environment that you install. Optional sub-features of the MVM Environment available are:

- **DOC** - Documentation Install Object includes child install objects if there are many kinds of documentation.
- **DPMI** - DOS Protected Memory Interface support.
- **EMS** - Expanded Memory Specification support.
- **XMS** - Extended Memory Specification support.
- **Sys Util** - System Utilities support includes, backup hard disk, change file attributes, display directory tree, manage partitions, label diskettes, link object modules, recover files, restore backed-up files, and sort filters.

WIN-OS/2 and Win32s installation is optional, and is dependent in the MVM server (with DPMI support) also being installed.

4.2.4 Multiple Virtual Machine Server

The MVM server in OS/2 Warp Connect (PowerPC Edition) provides the mechanism for the creation, initialization and destruction of Virtual Machines. A Virtual Machine is an environment in which a DOS, Windows or DPMI application can execute.

Although providing somewhat behind the scenes support, the MVM server is integral to the MVM environment. The server provides functionality that is similar to that of both the Virtual DOS Machine Manager (VDMM) and the MVDM Kernel found in OS/2 Warp (Intel).

The MVM server is closely integrated into the OS/2 Warp Connect (PowerPC Edition) environment. The MVM server starts via a RUNSERVER= entry in the CONFIG.SYS when OS/2 Warp Connect (PowerPC Edition) boots. Additional information on the OS/2 Warp Connect (PowerPC Edition) boot process is found in section 4.1.5, “Startup” on page 67. The MVM server provides the following services to the MVM environment:

- Identifies the hardware environment that it is running in (for example, the PowerPC) and loads the appropriate module binaries into the MVM environment.
- Loads and initializes the other components in the MVM environment by using the system loader in the task manager.
- Provides services to create, manage and destroy X86 VDMs in which DOS, Windows or DPMI applications can be executed. These services

are exported to other servers in the OS/2 Warp Connect (PowerPC Edition) service/personality layer in the form of a message based API.

- Exports a set of services to other components in the MVM environment as well as other components in the OS/2 Warp Connect (PowerPC Edition) service/personality layer to control the execution state of the Virtual DOS Machines.
- Provides a set of services to allow per VDM configuration to be done. This allows users to configure and control the resources associated or needed by the application that they need to execute in a particular VDM.

Although the MVM environment is structured as an OS/2 task it still utilizes the resources provided by the microkernel. It does this through the use of ports. A port is a unidirectional communication channel between a client that requests a service and a server that provides the service. Ports are discussed in more detail in section 2.1.3, “Inter Process Communication (IPC)” on page 10. In future releases of OS/2 Warp Connect (PowerPC Edition), the MVM environment will move away from the port communication structure and communicate directly to the OS/2 server.

The MVM server itself is comprised of a number of components. Many of these components provide similar services to those already found in the OS/2 Warp (Intel) environment. The MVM server components are:

- **Boot/Initialization**

  This component of the MVM server is the one that performs boot time initialization of the MVM environment.

- **Event List Management**

  The Event List Management component of the MVM server exports a set of services that are used by the emulation code (EM86 + VDDs) components in the MVM environment to register event handlers for various events.

- **Virtual Machine Manager**

  The virtual machine manager (VMM) supports services that control the operation of virtual machines though creation to termination. Access to the services are through the SERVICE_PORT.

4.2.5 EM86 (8086 Emulation)

The 80X86 emulation component (EM86) emulates 80X86 instructions and provides the functions required to support emulation to the other components of the MVM server. On an Intel based machine, EM86 would use the virtual 8086 mode of the Intel processors to provide low-level emulation functions. On the PowerPC, EM86 uses the services provided by the Instruction Set Translator (IST) to support the functions that it provides.

The EM86 components emulate some INT instructions, all IOPL-sensitive instructions, and some I/O instructions for 80X86 processors. These components also helps virtual device drivers emulate INT and I/O instructions. EM86 consists of the following major components:

- Boot-time initialization
- Client creation-time initialization
- Instruction emulation
- Virtual Device Helper (VDH) functions
- Hardware interrupt dispatcher.

EM86 uses the Instruction Set Translator (IST) to support the emulation of Intel instructions on the PowerPC. Low level instruction is performed by the IST. EM86 decodes instructions using the IST and passes control to the appropriate handler. If the instruction is unsupported or cannot be decoded, an invalid opcode (INT 6) is reflected to the VDM.

4.2.6 Instruction Set Translator

The Instruction Set Translator (IST) is a mapping layer to allow Virtual Device Drivers to use the current Virtual Device Helper (VDH) services with the IST. The IST also provides a *black box* emulator that emulates the Intel instruction set on non-Intel (PowerPC) hardware. This means that the Instruction Set Translator is the only area that understands Intel instructions on the non-Intel (PowerPC) machines.

The use and functionality of the IST is not restricted to the MVM environment. When the Intel Binary Support for OS/2 applications is added to OS/2 Warp Connect (PowerPC Edition) in a future release, the IST will be used to provide the required emulation support.

The architecture of the DOS emulation is such that the IST is perceived as a replaceable module. That is, if the MVM environment was moved to an Intel

based platform, then the function of the IST could quickly and easily be replaced by the actual Intel 80X86 processor.

The performance of the IST exceeds that of most of the other DOS/Windows software emulators that are available in the market today. In order to achieve this result it was necessary to limit the function supported by the IST. A brief summary of the limitations of the IST are listed below:

- Numeric coprocessor support is limited to 64 bits, instead of the 80 bits supported in the Intel Architecture.
- No limit checking for selectors.
- No checking of selector attribute bytes.
- No page table emulation (no page level memory protection or per-page memory management for protected mode applications). The means that it will not be possible to support the DPMI 0.95 demand-page memory functions.
- The IST is 386 only. Instructions specific for 486 (and above) are not supported.

4.2.7 DOS Emulation

The DOS Emulation Kernel emulates the DOS environment (such as DOS data structures and interrupt 21s) for each DOS session. DOS emulation is broken into two major pieces, one half that runs in the Intel address space (V86 mode on Intel or real mode in the IST) and the other half runs natively.

Figure 11 on page 81 shows the two parts of the DOS emulation servicing an interrupt in the MVM environment.

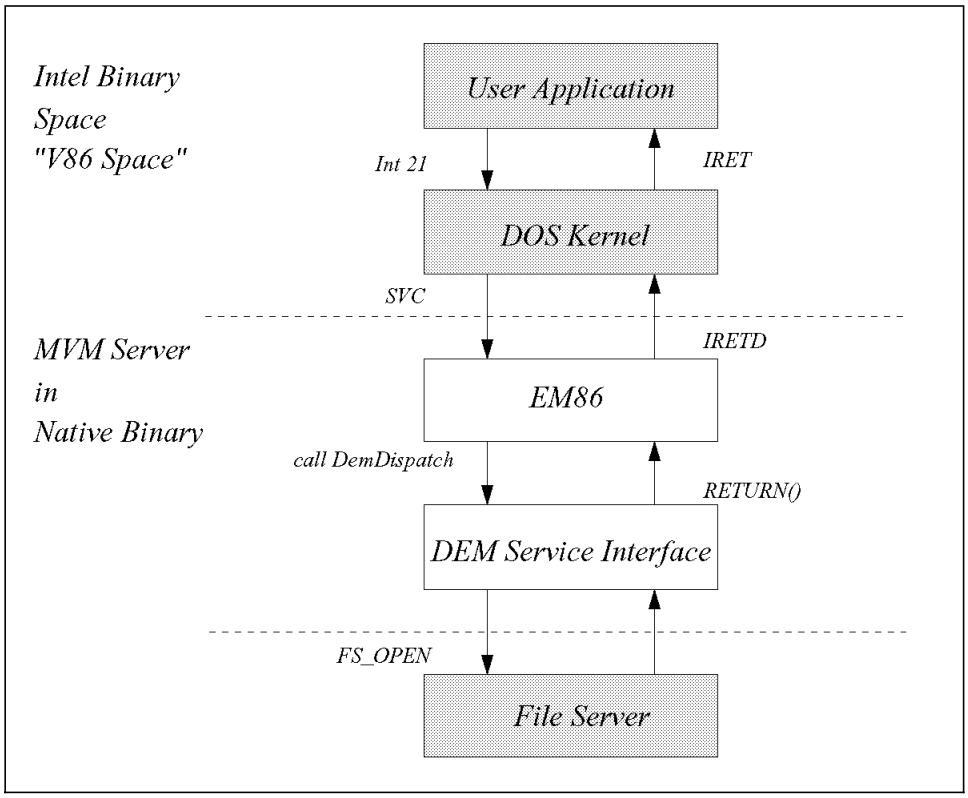


Figure 11. MVM Interrupt Processing

The piece of the DOS emulation running in the Intel space receives application requests in the Intel address space, updates the appropriate data structures, and then passes the request to the native piece of DOS emulation. The native piece of code acts of a router and routes the request to the appropriate worker routine, such as a file serve routine. The return code from the worker routine is then passed back to the Intel piece of code, so that it can return the proper result to the application.

The two components of the DOS emulation are:

- **DOS Kernel.** The DOS Kernel services the requirements of the DOS function calls that do not require services from outside the Intel space. An example of this would be the alloc_mem function call.
- **DOS Emulation Service Interface**. The DOS Emulation (DEM) Service Interface acts as a router of information requests for the DOS Kernel. The DEM interface is a PowerPC specific and is not present in the OS/2 Warp (Intel) version of the emulator. The DEM requests are routed to

either a Personality Neutral Service, a Cross Personality Service, the IBM Microkernel, a Virtual Device Driver, or a Physical Device Driver.

The services that the DEM layer provides include:

- **File I/O** - File, directory, and disk I/O to include Universal Naming Convention (UNC) names.
- **Device I/O** - I/O to devices to include IOCtl support.
- **Named Pipe I/O** - Support for the client end of named pipes.
- **Semaphores** - All semaphore support.
- **Internationalization** - Internationalization to support both DBCS and Unicode.
- **Other** - Other services such as initialization and termination support.

4.2.8 Virtual Device Drivers

DOS applications tend to access the hardware resources like devices directly. In order for multiple DOS applications, each running in a Virtual Machine, to access physical hardware devices, each virtual machine must be provided with a set of virtual interfaces to these devices. This is so the actions of one application running in a virtual machine does not affect the state of the device as perceived by other entities in the system.

The Virtual Device Drivers (VDD) provide these virtual interfaces. The VDD model used for the drivers is mostly unchanged since OS/2 Warp (Intel). The virtual device drivers that are supplied with OS/2 Warp Connect (PowerPC Edition) provide the same level of support that is available in OS/2 Warp (Intel).

Although the VDD model is mostly unchanged, there has been some changes to the implementation of the virtual device drivers in the OS/2 Warp Connect (PowerPC Edition) environment. One of the most obvious changes is that the virtual device drivers are no longer loaded at ring 0. Along with some of the physical device drivers, they are now at the user (ring 3) level of the operating system.

In OS/2 Warp (Intel), all of the virtual device drivers stored in the system as individual .SYS files and are loaded from the CONFIG.SYS file. In the first release of OS/2 Warp Connect (PowerPC Edition), that has changed. The base VDDs are found in the system dynamic link librariy MVDM.DLL, while the installable VDDs are still listed in the CONFIG.SYS file.

4.2.8.1 Available VDDs in OS/2 Warp Connect (PowerPC Edition)

The list of Virtual Devices Drivers in OS/2 Warp Connect (PowerPC Edition) is no different to that of OS/2 Warp (Intel), whether they are individual DLLs or combined in the MVDM.DLL file. Table 1 briefly defines the function of each of the available VDDs.

Table 1. OS/2 Warp Connect (PowerPC Edition) Virtual Device Drivers

VDD	Function
VAUDIO	Virtualizes Winos2 audio support
VBIOS	Virtualizes the BIOS data areas
VCDROM	Virtualizes CD-ROM support
VCMOS	Virtualizes the CMOS areas
VCOM	Virtualizes serial ports
VDISPLAY	Virtualizes Winos2 display (Uses OS/2 graphics DLL's)
VDMA	Virtualizes the DMA controller
VDPMI	Implements DPMI services
VDPX	Implements the DOS extender (DOSX) services for DPMI applications
VDSK	Virtualizes a hard disk controller
VEMM	Supports expanded memory (EMS) management
VFLPY	Virtualizes the floppy disk controller
VKBD	Virtualizes the keyboard controller
VLPT	Virtualizes the parallel ports
VMOUSE	Virtualizes the mouse
VNPX	Virtualizes the numeric coprocessor of Intel based machines
VPIC	Virtualizes the programmable interrupt controller
VTIMER	Virtualizes the Intel 8259A timer
VVIDEO	Virtualizes the video display
VW32S	Supports Win32s services
VWIN	Supports seamless Windows applications on the desktop
VXMS	Support extended memory management

4.2.9 Windows Support

The Windows support for the MVM environment is provided by WIN-OS/2. This is the same support that is currently provided in OS/2 Warp (Intel) The windows support provides Windows 3.1 application compatibility as well as support for Win32s version 1.15 and below applications .

The performance of the WIN-OS/2 subsystem is heavily dependent on the Instruction Set Translator (IST). On the PowerPC, all of the WIN-OS/2 code runs in the emulated Intel space, so the performance of the IST and the MVM server are pivotal to the performance that the Windows system will display.

By providing the same support for DOS and DPMI applications as OS/2 Warp (Intel), WIN-OS/2 is able to run unmodified for MVM. The OS/2 Warp Connect (PowerPC Edition) WIN-OS/2 implementation provides support for the Windows 3.1 API set (with the exception of VxD support).

4.2.10 Changes to The Command Set

OS/2 Warp Connect (PowerPC Edition) and the MVM environment, continue to offer the user a command line interface to the system.

In OS/2 Warp (Intel), due to the similarity of the OS/2 and DOS versions of these utilities, many of the utilities were coded using the Family API (FAPI) so that a single binary executable resided on the system.

With the lack of 16 bit OS/2 support in the current release, FAPI is no longer an option. However, by splitting the utilities into separate executables, the DOS versions can offer a flexibility that was unavailable in the previous OS/2 releases. Future releases of OS/2 Warp Connect (PowerPC Edition) will allow DOS only devices such as network redirectors and other block device drivers. In addition, the operational level of the DOS compatible function can be improved beyond what real DOS is capable of. The START command, or the seamless integration with OS/2 and Windows programs can be done in a superior fashion as compared with similar offerings in the marketplace.

The DOS functionality for the utilities has been taken from PCDOS 6.3.

For this reason, and other architecture considerations, several OS/2 and DOS commands and utilities are no longer available in OS/2 Warp Connect (PowerPC Edition). The changes to the available utilities are shown in Table 2 on page 85.

Table 2 (Page 1 of 2). Changes to DOS Utilities

Utility Name	Comments	Added	Deleted
ARABIC	A new command to OS/2 Warp Connect (PowerPC Edition), it provides for ARABIC character support in VDM sessions.	√	
BOOT	Since OS/2 and DOS are not available on the PowerPC platform as native systems there is nothing to dual-boot to.		√
CACHE	Caching has been reimplemented in OS/2 Warp Connect (PowerPC Edition), making this command no longer required.		√
CHOICE	This command is used in batch files to display a specified prompt. The prompt provides the opportunity to choose whether or not the batch program is to be processed.	√	
CRC	Gives a Cyclic Redundancy Check (CRC) number for a filename. The number can be used to uniquely identify a file. An OS/2 equivalent function also exists in OS/2 Warp Connect (PowerPC Edition).	√	
CREATEDD	New System Management routines with regard to dumps and dump taking, has removed the need for this command.		√
DELTREE	Allows users to delete whole directory trees from a DOS command line. There is no equivalent OS/2 command line function.	√	
DRVLOCK	Locks or unlocks the specified drive or socket.	√	
E	A full screen text editor for DOS.	√	
EDLIN	This editor was replaced with the DOS E editor.		√
HELPMSG2	The help system has been re-worked make this file redundant.		√
NLSFUNC	The internationalization in OS/2 Warp Connect (PowerPC Edition) has replaced this function.		√
REXX	Added REXX support for DOS sessions, similar to that found in PC DOS 7.0. This change allows REXX program execution in both DOS and OS/2 sessions in OS/2 Warp Connect (PowerPC Edition).	√	

Table 2 (Page 2 of 2). Changes to DOS Utilities

Utility Name	Comments	Added	Deleted
RC	Not supported due to architecture change.		√
SETBOOT	This utility is part of a multiboot system, which is not an OS/2 Warp Connect (PowerPC Edition) Release 1 function.		√
SETVGA	Not supported in this release.		√
SHARE	Added DOS share.exe support to VDM sessions in addition to support already provided by OS/2.	√	
START	This utility allows users to start new sessions from the command shell. This function was restricted to OS/2 sessions in OS/2 Warp (Intel).	√	
SVGA	Not supported in this release.		√
SYSLEVEL	Allows the user to check the syslevel of the DOS components.	√	
VIEWDOC	Functionality has been rolled into the VIEW command.		√

In addition, due to the lack of FAPI support in OS/2 Warp Connect (PowerPC Edition), several more utilities are now found in the MDOS directory. These commands perform the same function as they did previously in OS/2 Warp (Intel), and have native OS/2 equivalents. They are:

- CHKDSK
- EAUTIL
- PATCH
- UNDELETE

4.2.11 Changes to the MVM DOS Settings

The DOS settings that are used to configure MVM sessions in OS/2 Warp Connect (PowerPC Edition) are very similar to those available to current users of OS/2 Warp (Intel). However, several changes have been made, with new DOS settings appearing in OS/2 Warp Connect (PowerPC Edition), and others being no longer supported. A summary of these changes are shown in Appendix A, “Changes to MVM DOS Settings” on page 143.

4.3 Graphics Subsystem

Graphics Subsystem is a part of OS/2 Warp Connect (PowerPC Edition) that is responsible for drawing all graphic requests from applications to the specific hardware output.

The Graphics Subsystem exists in two components of OS/2 Warp Connect (PowerPC Edition) as shown in Figure 1 on page 2:

- Presentation Services
- System Services

For detail information about Graphical Subsystem in OS/2 Warp Connect (PowerPC Edition), please refer to document, *OS/2 Warp Connect (PowerPC Edition), Graphical Subsystem*, SG24-4639.

A Part of graphics subsystem that resides in Presentation Services are:

- Graphics Engine
- Translation Layer (GRE2VMAN, GDI2VMAN)

Graphics Engine will be discussed in 4.3.2, “Graphics Engine” on page 89. Translation Layer is a part of GRADD Model that will be described in 4.3.3, “PM Video Device Driver” on page 92.

The rest of the graphics subsystem that resides in System Services are:

- VMAN
- GRADD
- Softdraw
- Base Video Services

VMAN, GRADD and Softdraw are part of Graphics Subsystem which will be discussed in 4.3.3, “PM Video Device Driver” on page 92. Base Video Services will be discussed in 4.3.4, “Base Video Services” on page 98.

4.3.1 Graphics Subsystem Overview

This section will describe the overview of the new model that is specifically designed for OS/2 Warp Connect (PowerPC Edition).

The Graphics Subsystem of OS/2 Warp Connect (PowerPC Edition) provides the same functions as OS/2 Warp. There are changes and enhancements due to Microkernel architecture.

The new video device driver can be written by the developer using Graphics Adapter Device Driver (GRADD) APIs. GRADD will simplify the device driver by providing a smaller number of mandatory functions to be implemented.

The structure of OS/2 Warp Connect (PowerPC Edition) Graphics Subsystem is shown in Figure 12.

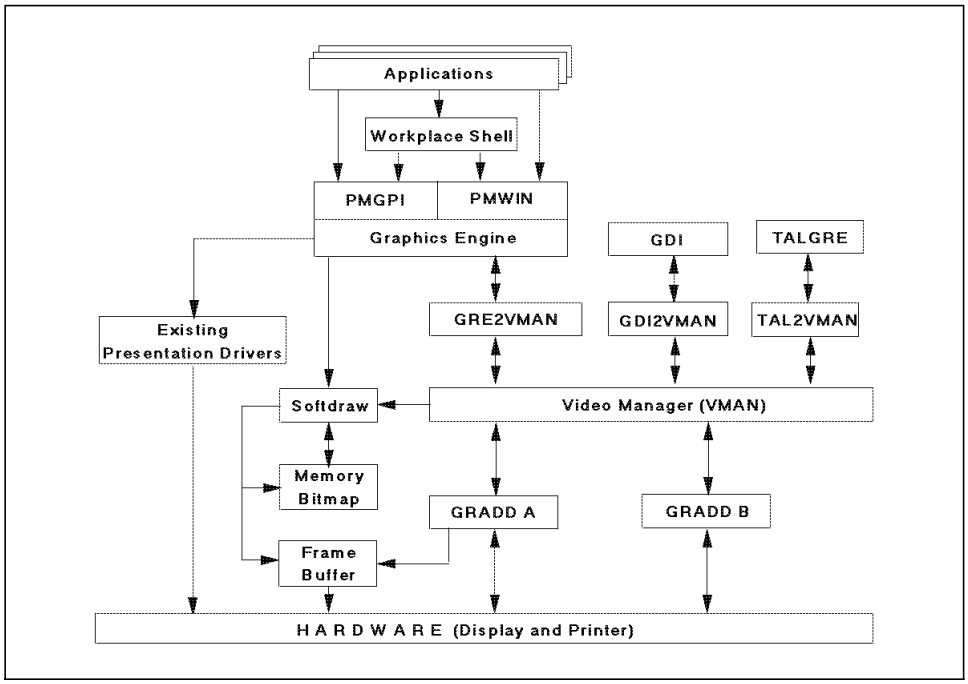


Figure 12. OS/2 Warp Connect (PowerPC Edition) Graphics Subsystem

PMGPI The Graphics Programming Interface provides the means used by applications to do graphic requests. For example, to draw an arc and/or to write a text at a certain position on the screen, an API call is made to PMGPI.

PMWIN The Window Manager is responsible for creating, maintaining and destroying windows on the PM desktop. For example, if we want to open the pop-up dialog, the mechanism will be provided by PMWIN.

PMGRE The PM Graphics Engine is the core of PM System. It will be called by PMWIN and PMGPI on behalf of the applications. It works very closely with the device driver.

PDs Presentation Drivers are the device-dependent tools used by Graphics Engine to map its graphics layout. Presentation Drivers will be different for every hardware supported.

Softdraw Softdraw stands for Software Drawing for Non-Accelerated Graphic Operation. Softdraw will be used by Graphics Engine when it needs to do some raster graphics. Softdraw is needed to perform raster operations to a linear address space.

GRE2VMAN The GRE2VMAN is also called a translation layer. It is responsible for reporting current GRADD modes and capabilities to the Graphics Engine. GRE2VMAN is the first translation layer that will be loaded if the system uses PM as the dominant personality.

VMAN Video Manager is the main component of GRADD model that will be responsible for serializing the communication between access to the GRADDs and the translation layer. Video Manager can also call Softdraw for simulation.

GRADD Graphics Adapter Device Drivers provides video support to all the graphics subsystems which can run on the OS/2 Warp Connect (PowerPC Edition). A GRADD contains only the hardware dependent code that is needed to graphic functions that are common among the different graphics subsystems.

4.3.2 Graphics Engine

This section will discuss GRADD model as a new graphics subsystem model in OS/2 Warp Connect (PowerPC Edition).

Figure 13 on page 90 shows the detail structure of OS/2 Warp Connect (PowerPC Edition) Graphics Engine.

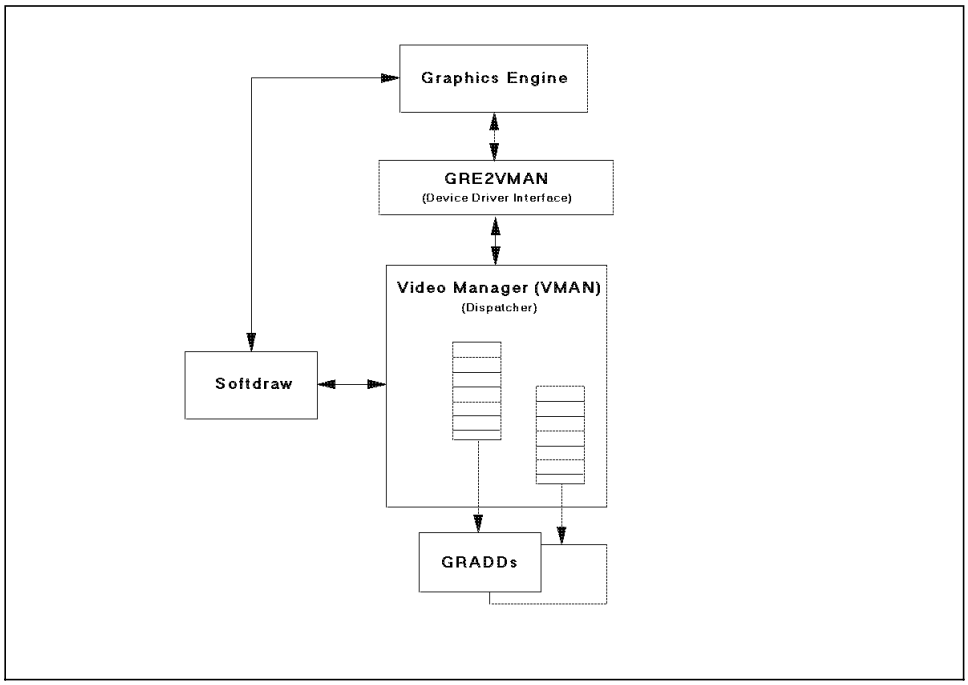


Figure 13. OS/2 Warp Connect (PowerPC Edition) Graphics Engine

Graphics Engine (PMGRE) will receive and forward requests for graphic operations. The capability of Graphics Engine in OS/2 Warp Connect (PowerPC Edition) is enhanced and some mandatory functions are added. These enhancements will be described later in this section.

The Device Driver Interface function of OS/2 Warp Presentation Driver has been changed by *GRE2VMAN*. The GRE2VMAN is also called a translation layer in GRADD model. It is responsible as a callpath between Video Manager (VMAN) and Graphics Engine. It is also responsible for handling the OS2_PM_ENABLE and DevEscape functions.

GRE2VMAN will then forward the requests to *Video Manager (VMAN)* module which will handle serialization and dispatching to the correct device driver. VMAN is part of Shared Services.

The display device driver for OS/2 Warp Connect (PowerPC Edition) is now called *Graphics Adapter Device Driver (GRADD)*. GRADD can now perform the function to the specific hardware via Microkernel but it can also return RC_SIMULATE which will allow VMAN to call SOFTDRAW for simulation.

Softdraw fully performs rasterization functions. In the previous version of OS/2 it can be performed also by Presentation Driver.

Table 3 is the summary of OS/2 Warp Connect (PowerPC Edition) Graphics Engine Structure.

Table 3. A Summary of OS/2 Warp Connect (PowerPC Edition) Graphics Engine

No.	Module	Task
1.	PMGRE	Translates PM graphic primitives into PMGRE independent GRADD primitives.
2.	GRE2VMAN	Converts the commands and send them to VMAN using Video Manager Interface (VMI) as a protocol.
3.	VMAN	Responsible for serialization of GRADD updates. commands to GRADD using Graphics Hardware Interface (GHI) as a protocol.
4.	GRADD	Performs the operation or returning RC_SIMULATE to inform that the commands need to be simulated. VMAN then calls SOFTDRAW.
5.	SOFTDRAW	Simulates the commands as requested by VMAN and also fully performs rasterization functions.

The GRADD model will benefit the device driver developer since it needs a small number of functions that need to be implemented before a system can be fully functional.

The structure of OS/2 Warp Connect (PowerPC Edition) Graphics Engine not only gives the easier way to enhancing the device driver but it also reduces the time for maintenance of the Presentation Driver (PD), as shown in Figure 14 on page 92.

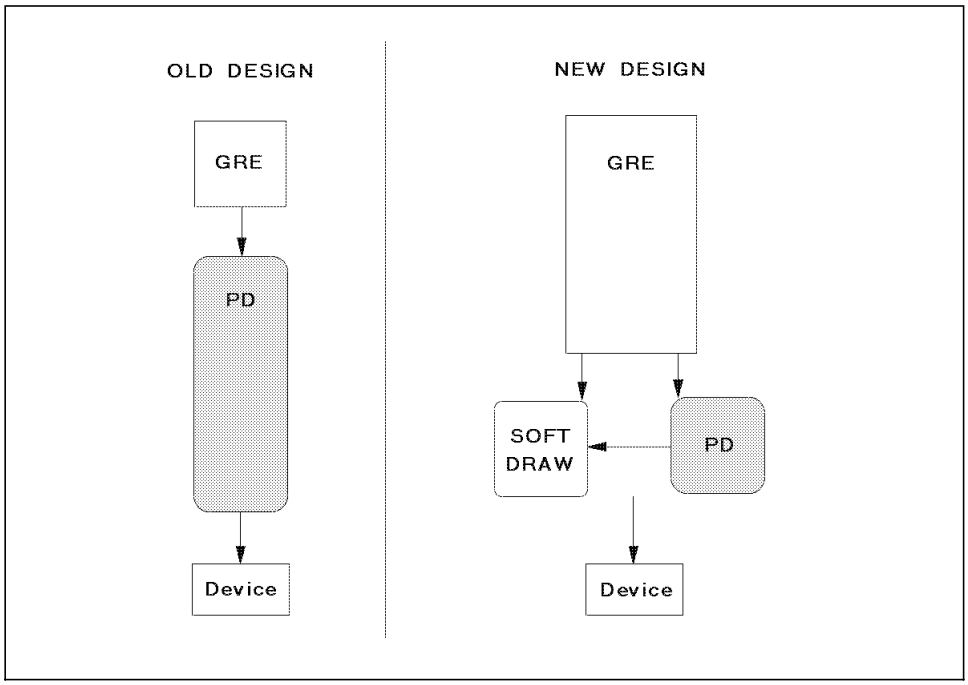


Figure 14. GRE/Presentation Driver Design

The Graphics engine is being designed so that a developer can create PD with minimal effort, and then incrementally add functions that support specific hardware. In order to achieve this, the new design will:

- Have the GRE manage contextual/state information
- Provide a copy of contextual/state information when the PD hooks functionality
- Provide a device contextual serialization routine
- Perform rasterization into a linear address specified by the PD
- Provide a set of device support routines

4.3.3 PM Video Device Driver

This section will describe the PM Video device driver model for OS/2 Warp Connect (PowerPC Edition).

The PM Video device driver model for OS/2 Warp Connect (PowerPC Edition) incorporates two models:

- Existing PM Video device driver model for OS/2 Warp

- New model called the Graphics Adapter Device Driver (GRADD) model

OS/2 Warp Connect (PowerPC Edition) will continue to support the existing full Device Driver Interface, as described in the OS/2 2.1 Presentation Driver Reference. Existing OS/2 2.X drivers that are being ported to OS/2 Connect (PowerPC Edition) will not be forced into using the GRADD model, although it will be the recommended approach for driver developers from now on.

The *GRADD* model design is a clean and simple architecture, giving developers the opportunity to write display device drivers very easily and quickly. The GRADD model is devided into several components that coordinate the communication between OS/2 and the available hardware. These components are:

- *Translation layers* (GRE2VMAN, GDI2VMAN, TAL2VAM, etc.)
- *Virtual VMI VDD (VVMI)*
- *Video Manager (VMAN)*
- *Graphics Adapter Device Drivers (GRADDs)*
- *Softdraw*

The GRADD model with all its components is shown in Figure 15 on page 94.

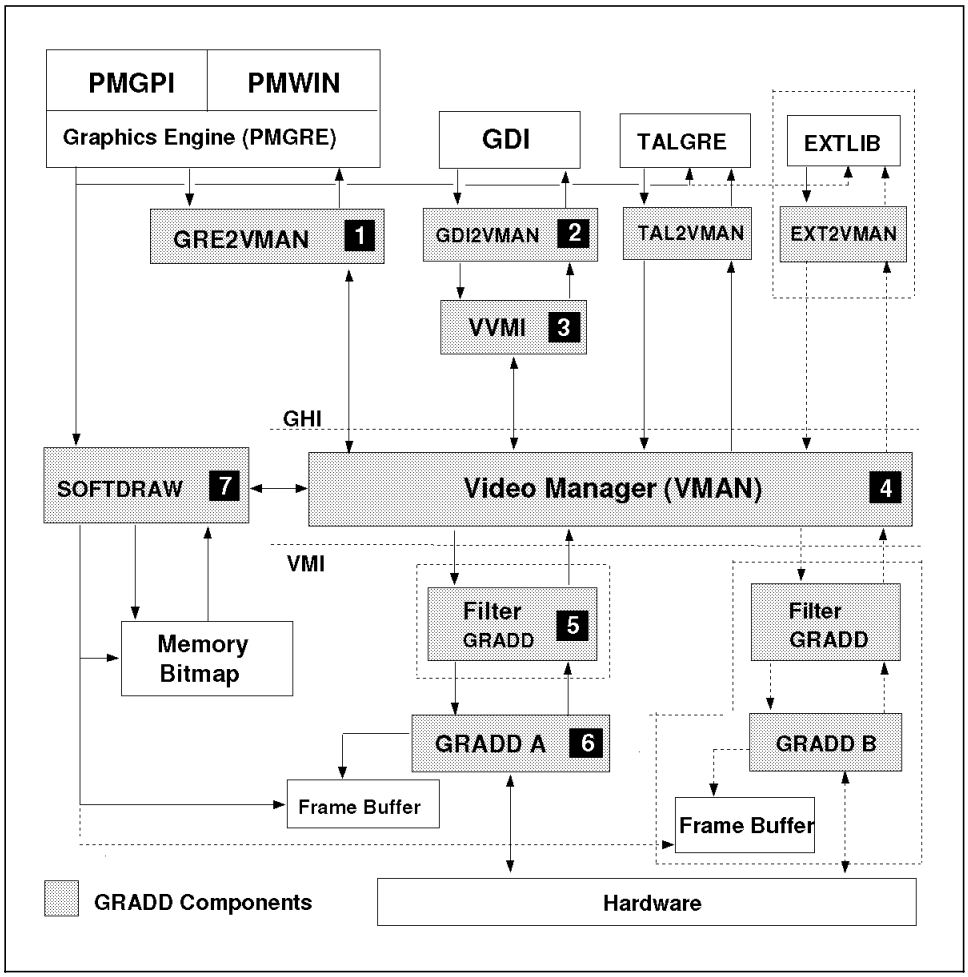


Figure 15. GRADD Model

The following lines explain how these components handle a basic operation:

- The translation layer sends VMAN one of the defined Video Manager Interface (VMI) commands.
- Upon receiving a VMI command, VMAN waits for a semaphore, if required. This semaphore is used to protect the hardware from more than one thread accessing a GRADD, and therefore protecting the hardware at the same time.
- VMAN then sends the hardware command to the appropriate GRADD.

- After the GRADD has received the Graphics Hardware Interface (GHI) command which is not mandatory, it has the option of performing the request operation or returning the request back to VMAN with a return code of RC_SIMULATE. The RC_SIMULATE informs VMAN that the command needs to be simulated in software.
- VMAN then calls a component Softdraw to simulate the operation.

The following pages will discuss each component in the GRADD model.

1 *GRE2VMAN* is the first translation layer and the first component of the GRADD model to be loaded and called by the GRE. When GRE2VMAN is loaded, it calls VMAN's VMIEntry function with a VMI_CMD_INIT command. When VMAN receives a command VMI_CMD_INIT for the first time, it loads the other GRADD model components.

GRE2VMAN.DLL translates the PM graphics' engine commands into VMI commands. GRE2VMAN is a passthrough from PMGRE to VMAN. PMGRE has also been optimized to package the drawing command before calling GRE2VMAN. This technique reduces the number of calls down to the video subsystem and helps overall performance.

The GRE2VMAN component will also be responsible for handling the OS2_PM_ ENABLE and the DevEscape functions.

2 *GDI2VMAN* is a Windows translation layer. It translates the Windows graphics engine (GDI) commands into VMI commands. GDI2VMAN calls VMAN directly via a virtual device driver called VVMI (Virtual Manager Interface). Windows support is accomplished this way.

3 The *Virtual VMI VDD (VVMI)* component provides only a callpath from GDI2VMAN to VMAN.

4 *Video Manager (VMAN)* is the primary component in the GRADD model, as shown in Figure 15 on page 94. VMAN is responsible for the following:

- Communication
- Input Management

VMAN is a link between the translation layers and the GRADDs and is used to serialize the communication between these components.

VMAN relies on a special protocol to receive requests from the translation layers. The VMAN protocol is called the Video Manager Interface (VMI). This protocol consists of small set of operations which are individually

identified by a function number. The translation layers communicate to VMAN via one entry point using export functions.

For input management, mouse movement is sent from the Event Window Server (EWS) to the Video Manager (VMAN). The mouse event thread calls VMAN which calls the GRADD to perform a mouse pointer update. The GRADD has the options to update the pointer or return to VMAN for simulation. When RC_SIMULATE is returned, then VMAN will simulate the pointer movement using the regular BitBlt commands.

5 The *Filter GRADD* component is shown in Figure 15 on page 94. This component is optional in the GRADD model. There is no way anyone can anticipate the changes in the graphics' hardwares. We must allow a mechanism to extend the architecture in a manner which takes fullest advantage of the new hardware.

The Filter GRADD provides a way of modifying the GRADD's behavior without rewriting and compiling the GRADD. A Filter GRADD is placed between VMAN and the GRADD. This component provides now an extra link to the GRADDs. This is called chaining.

The GRADD model can be extended by using the VMI_CMD_EXTENSION command. An extension can be written to pass its own defined commands to a GRADD or a new GRADD can be written to handle the additional support for a given extension.

6 The *GRADD* as shown in Figure 15 on page 94 is a hardware specific device driver. GRADDs contain only a small set of common functions, which are designed to act as building blocks for the larger more complex operations. These complex operations are required by the GRE.

GRADDs are the only components in the model that have direct access to the hardware. A GRADD contains only the hardware specific code to exploit the accelerated features of a specific graphics adapter. For most developers, the GRADD will be the only code that needs to be written.

GRADD relies on a special protocol to receive requests from VMAN. The GRADD protocol is called Graphics Hardware Interface (GHI). A GRADD receives all of the operations from VMAN via a single export function called HWEntry. HWEntry is the exact same as the VMIEntry.

7 *Softdraw* is the component in the GRADD model, as shown in Figure 15 on page 94, that, provides the Software simulation. Softdraw consists of a generic graphics library to be used by VMAN and the system for software

simulations. Softdraw exports two functions called SDBitBlt and SDLine that are used by VMAN to simulate in software bitblt and line operations, respectively. Softdraw can draw the bits directly into the bitmap, if given a pointer to a linear address, a VRAM bitmap or system bitmap. The Softdraw component will support rasterization concepts.

Software simulation allows a developer to write the driver in incremental stages, rather than writing the entire driver up front. Once the mandatory functions are completed, a developer can use the software simulation to simulate the non-mandatory functions. When the non-mandatory functions are coded to exploit the capabilities of the graphics adapter, the result can be compared to the result of the software simulation. This is a way for the developers to assure that their PM drivers look consistent to drivers written for different hardware platforms.

The PM Video Device Driver model for OS/2 Connect (PowerPC Edition) is shown in Figure 16 on page 98. Note the Graphics Engine will either dispatch drawing requests to a driver coded to the existing DDI interface or to the new driver coded to the GRADD model via the Video Manager(VMAN) interface.

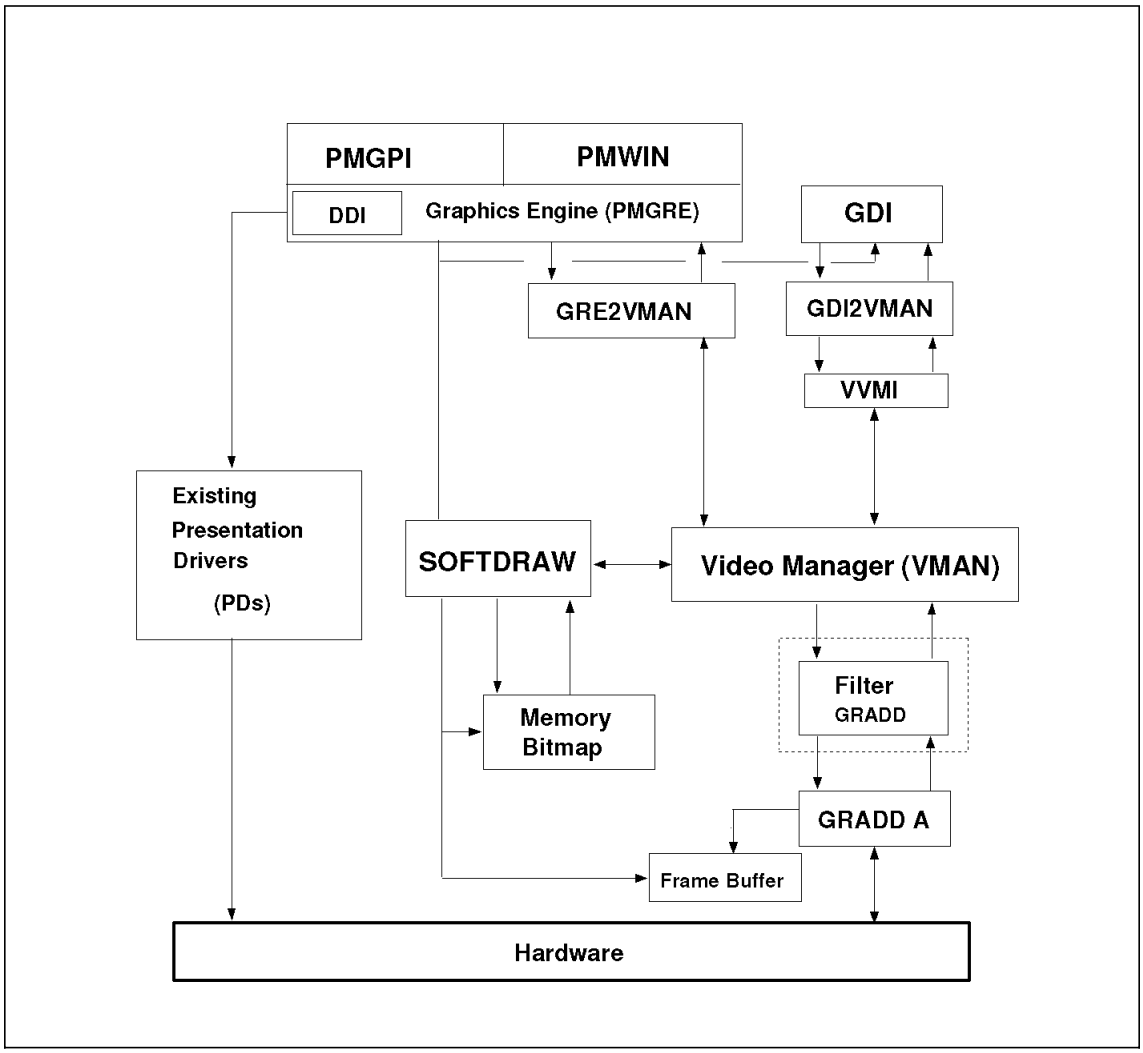


Figure 16. OS/2 WARP Connect (PowerPC Edition) PM Video Device Driver Model

4.3.4 Base Video Services

The Base Video Subsystem is comprised of a shared module (VIDEOPMI) which communicates to/from the protected mode video device driver (BVHSVGA - OS/2 Warp Connect (PowerPC Edition)), as well as the virtual video device driver (VSVGA - Multiple Virtual Machine).

It is the part of Presentation Manager which maintains compatibility with text mode application and graphical system in protected mode. The discussion of

Base Video Subsystem in OS/2 Warp Connect (PowerPC Edition) will deal with the design and function of VIDEOPMI.

VIDEOPMI is a neutral subsystem which provides services to:

1. BVHSVGA for OS/2 full-screen sessions
2. BVHWNDW for OS/2 windowed session
3. VSVGA for VDMs
4. VIO API for 32-bit programming interface
5. Base Video Subsystem

All these callers will be discussed separately in 4.3.4.1, “Text Mode in OS/2 Warp Connect (PowerPC edition)” on page 101. Virtual Video Device Driver which provides DOS Full screen and windowed support will be discussed in 4.3.4.2, “ Virtual Video (VVIDEO)” on page 103. VIDEOPMI will be called by GRADD when an application requests for example, drawing a line in a OS/2 windowed. The structure is shown in Figure 17.

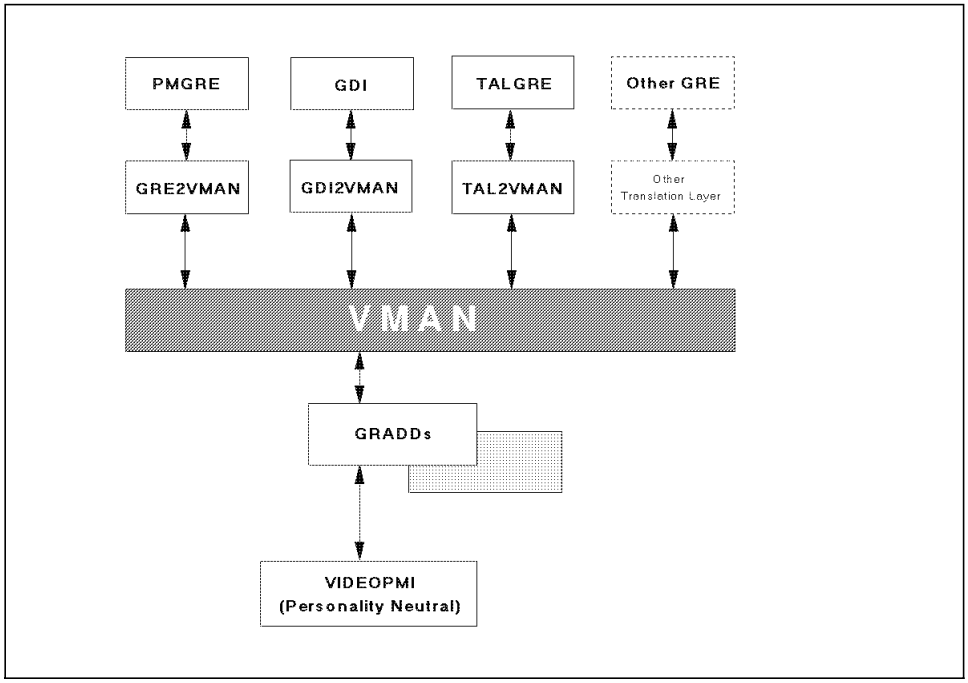


Figure 17. VIDEOPMI Accessed from GRADD

The design of VIDEOPMI is built around information contained in the Protect-Mode Interface (PMI) file. It is based on the SuperVGA Protect-Mode Interface (SVPMI) VESA standard.

The PMI file will contain data and commands necessary to provide support for modes beyond VGA in a non-BIOS environment. Adapter manufacturer or display device driver provider is responsible for providing PMI file.

The main goals of VIDEOPMI are:

- To centralize all of the setmode related services
- To provide a consistent interface which is not dependent on any BIOS services
- To facilitate multiple adapter support and dynamic video configuration

In its implementation, VIDEOPMI will be called by the callers. All functions will request one entry point. All calls are then routed to the appropriate routine. VIDEOPMI will not hold any instance data, all data must be maintained by the caller. All data maintained by VIDEOPMI will be allocated dynamically as shared data during initialization. And since the data will be shared, it will need a handle which is the base address of data allocated during initialization. This handle will be notified and referenced to every caller.
Initialization of VIDEOPMI takes the following four main steps:

1. Allocating and initializing internal memory management routines
2. Loading and parsing of the PMI file
3. Setting up addressability to code and data
4. Notifying the Virtual Video Driver of entry points and handle values

The most important file in VIDEOPMI is Protect-Mode-Interface file. This file resides in directory \OS2. The first release of OS/2 Warp Connect (PowerPC Edition) provides 3 PMI files: WD_90C24.PMI, S3_864.PMI and S3_928.PMI. These files are readable and will be parsed by VIDEOPMI Initialization and put in shared memory as an internal linked-list.

The PMI file is responsible for:

- Describing a graphics adapter to the OS/2
- Providing means to set, save and restore video modes in Protected-Mode
- Providing parameters for the adapter virtualization in multiple DOS session.

The PMI language and its interpreter can facilitate dynamic hardware configuration, which includes port remapping, adding or removing an adapter and its PMI definition, changing the attached display and multiple instances of the same video hardware. It also facilitates different refresh rate support and programming of external video-support chips, such as clock chips or smart DACs. PMI file should list default adapter port configuration and mode sections which are capable of supporting different refresh/monitor setups. Monitor configuration, port mapping, and selection of the adapter instance are performed by components other than VIDEOPMI.

The PMI file can list one or more adapter description starts with its hardware section, followed by IdentifyAdapter and a number of support sections and a list of all available modes with the actual hardware sequence required for a successful mode set from any hardware state of the adapter. If multiple adapters are listed, support sections for each adapter are considered private and cannot be referenced from other adapter descriptions.

PMI files are to be provided by the video chip or adapter manufacturers, or by the providers of the display drivers. The file should be part of the video adapter installation kit, either as a pre-manufactured flat file or one just created by the OEM′s installation utility. OS/2 Warp Connect (PowerPC Edition) Installing utility will add the file into the OS2.INI file with its full pathname and with adapters OEMString identifier.

PMI services for a recently installed adapter are available as soon as the hardware described in its PMI file is installed and available. The adapter is considered available if IdentifyAdapter section in the PMI file returns true.

4.3.4.1 Text Mode in OS/2 Warp Connect (PowerPC edition)

The need for the VIO and Keyboard calls has not gone away, and to protect legacy applications they must continue to be supported. There also remains the needs for a simple program video and keyboard interface. While PM gives a rich graphical interface, it is difficult to write a simple PM program just for simple requirements for keyboard and screen output.

OS/2 Warp Connect (PowerPC Edition) has to provide a simple text mode interaction. In OS/2 Warp Connect (PowerPC Edition) there is a legacy of VIO APIs and in DOS there is a legacy of BIOS and direct hardware I/O.

OS/2 Warp Connect (PowerPC Edition) provides base video support by implementing 32-bit versions of most of the OS/2 1.x VIO calls, while for DOS applications Multiple Virtual Machine (MVM) provides textmode support using VDDs. More discussion on MVM can be read in 4.2, “The MVM Environment” on page 72.

Since the current APIs are 16 bit only, binary compatibility with existing applications is not possible. However, in most cases it is possible to move to the 32-bit VIO APIs by doing only a recompile.

The concept in OS/2 Warp Connect (PowerPC Edition) provides a simple graphic interface based on a generic monospaced rectangular text window where each character has an attribute. This can be implemented on any hardware base. OS/2 Warp Connect (PowerPC Edition) has an integrated OS/2 Base Video Handler and Presentation Manager so that all user interaction in OS/2 goes through Presentation Manager.

Full screen sessions are in fact just PM Windows with no border. PM can then decide whether to use a graphical mode or hardware text mode to actually display them. Both DOS and OS/2 sessions are then movable between windowed and full screens. This requires that the font be scaled to match the size of the physical screen (unlike the normal maximize).

The OS/2 text mode support (OS2CHAR) provides a common set of video code to implement both full screen and windowed VIO. OS2CHAR is a component of the OS/2 Presentation Manager which processes input for OS/2 sessions. OS2CHAR is maintained by a per session logical video buffer.

The base video handlers are implemented as VIDEOPMI which is documented in 4.3.4, “Base Video Services” on page 98. These are called from the OS/2 and MVM to do mode set, save, and restore.

The BVS (Base Video Services) and BVH (Base Video Handler) layers no longer exist. Most of the functions have been moved to VIDEOPMI, and the rest is moved directly into the OS2CHAR implementation. The consequences are that all video presentation drivers must support the small number of AVIO calls. In OS/2 Warp Connect (PowerPC Edition) implementation, these functions are implemented by the graphics engine.

The structure of Text mode processing in OS/2 Warp Connect (PowerPC Edition) will be basically the same as Figure 17 on page 99 since OS2CHAR is only the part of PM. The more detail structure including MVM (DOS) module is shown in Figure 18 on page 103.

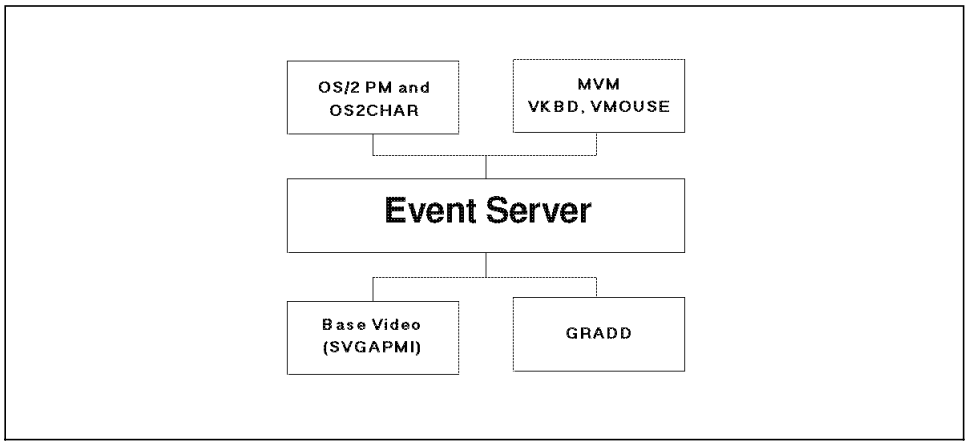


Figure 18. OS/2 Warp Connect (PowerPC Edition) Text Mode

Event Server is a part of Event and Window Services (EWS). It runs as a thread which is responsible for processing the event input port. Event input port is a microkernel port which receives messages from the interrupt logic for the keyboard and mouse devices. More discussion on EWS could be seen in 3.2, “Event and Window Services” on page 32.

Return codes from the VIO calls are changed from 16-bit API (USHORT) to 32-bit API (APIRET) and most USHORTs in the parameter lists have been changed to ULONG.

4.3.4.2 Virtual Video (VVIDEO)

Multiple Virtual Machine (MVM) Environment is the part of the OS/2 Warp Connect (PowerPC Edition) operating system that is responsible for providing the DOS/WINDOWS/DPMI program compatibility. Virtual Video (VVIDEO) is one component of MVM that is responsible for routing all the video requests to hardware.

The VVIDEO component processes INT 10h requests for video events from the v86 thread and services them. Virtual 8086 mode is historically a component of 80386 intel chip which enables system software to emulate an 8086 environment with a *virtual machine*. In PowerPC, it will use 8086 Emulation Component (EM86) which is part of MVM.

The VVIDEO exists in three parts:

- I/O Thread
- INT 10h Emulator

It hooks interrupts generated by the DOS application. It also requests query or set the video state from the v86 thread and services them.

- Port Handler Routines.

  It services *In* and *Out* requests for the video ports. It also hooks all relevant video ports.

The VVIDEO consists of the following components:

1. **Boot-time Initialization**

   This component performs the global initialization at the MVM server boot initialization time and functions as follows:

   - Initializes the global data structures
   - Registers the user handler VVCreate with the MVM server and must be called every time a new VDM is created

2. **Client Creation-time Initialization (VVCreate)**

   The client creation-time initialization component is called every time a new virtual DOS machine (VDM) is created. The VVCreateVM function is called in the context of the newly created VDM and does the following:

   - Installs INT 0x10 hook handler, VVINT10Hook
   - Installs video port handler, VVIOHookHandler for the device dependent ports
   - Communicates with the SVGAPMI PNS to gain port rights to the video hardware

3. **Software Interrupt and port I/O emulation**

   Instruction emulation is provided through the Instruction Set Translator (IST). IST is responsible for emulating Intel instructions on non-Intel machines, such as the PowerPC. It provides 386 ring 3 protected mode and real mode. IST is the only module that understands Intel instructions, and provides services that the rest of MVM depend upon to run DOS and Windows applications on a PowerPC. I/O port requests will be passed on to virtual video provided that the virtual video driver has called VDHInstallIOHook. Conversion of Intel based port I/O to the appropriate memory mapped load-and-store will accomplished in the IST.

Figure 19 on page 105 provides a general view of the virtual video device driver architecture.

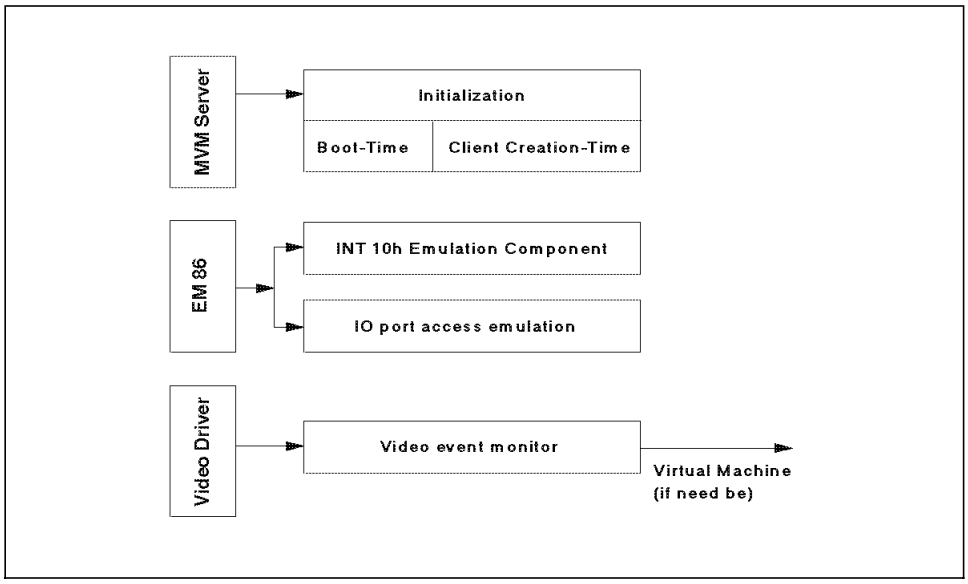


Figure 19. VVIDEO Internal Architecture

For optimal performance and compatibility, the Video VDDs support full-screen operation. For convenience, an option appears on the VDM system menu to convert the VDM from Windowed mode to Full-screen mode and back.

The Video VDDs support full-screen operation by doing the following operations:

- Registering foreground/background screen-switch notification hooks with the VDM Manager
- Utilizing the save/restore video buffer services which is provided in the PNS of base video
- Installing I/O hooks to shadow key video port accesses
- Mapping physical video memory into the appropriate video address space
- Coercing text mode fonts to match the currently selected codepage
- Providing pointer drawing services to the mouse VDD to define, draw and erase pointer images

4.3.4.3 Virtual Windows (VWIN)

Virtual Windows (VWIN) is a virtual device drive that allows a Windows program to run on the OS/2 2.x desktop, side by side with OS/2 applications and other Windows applications. It is the link that passes messages from one GUI to another so that both environments can know about and adjust for each other.

In the first release of OS/2 Warp Connect (PowerPC Edition), seamless will work the same, but the components will be implemented differently to fit the OS/2 Warp Connect (PowerPC Edition) Architecture.

In OS/2 Warp Connect (PowerPC Edition), VWIN will provide a number of services to facilitate the communication between Win-OS/2 and the presentation task. Win-OS/2 will use INT 66h to access it. VWIN will exist in each VDM and do the following tasks:

- Receive all messages from presentation task
- Send all Win-OS/2 messages

The presentation task will use APIs, generated with MIG (Message Interface Generator), to send and receive messages from Win-OS/2.

The exchange of messages between the VDM task and the presentation task represents a peer-to-peer form of client/server communication. At one point, the VDM may be the server, handling messages sent from another VDM or the presentation task. At another point, the presentation task may become the server, handling requests from the VDMs.

In OS/2 Warp Connect (PowerPC Edition), WinShield will call into VWIN as it does in OS/2 Warp by executing an INT 66h, causing the Interrupt Handler to execute the VWIN entry point function for Win-OS/2. VWIN in VDM needs send rights to the port of VWIN in presentation task in order to send messages.

To receive a message, a stub routine will exist that listens at VWIN′s receive port, waiting for incoming messages. Multiple threads will exist to receive the messages. They are threads to receive shield messages, error messages and clipboard/DDE messages. Once a message arrives, it will be placed on a local message linked list. WinShield can then access the list through an INT 66h call to VWIN.

The *VWIN Central Services Task* will be started by the presentation task VWIN code. The following list are the tasks of VWIN Central Services Task:

- Handles all clipboard/DDE requests
- Holds the *master* clipboard
- Contains a list of all VDMs and the presentation task
- Routes the broadcast messages from the VDM or Presentation Task using a list of all VDMs and the presentation task

Overall message flow of OS/2 Warp Connect (PowerPC Edition) can be seen in Figure 20.

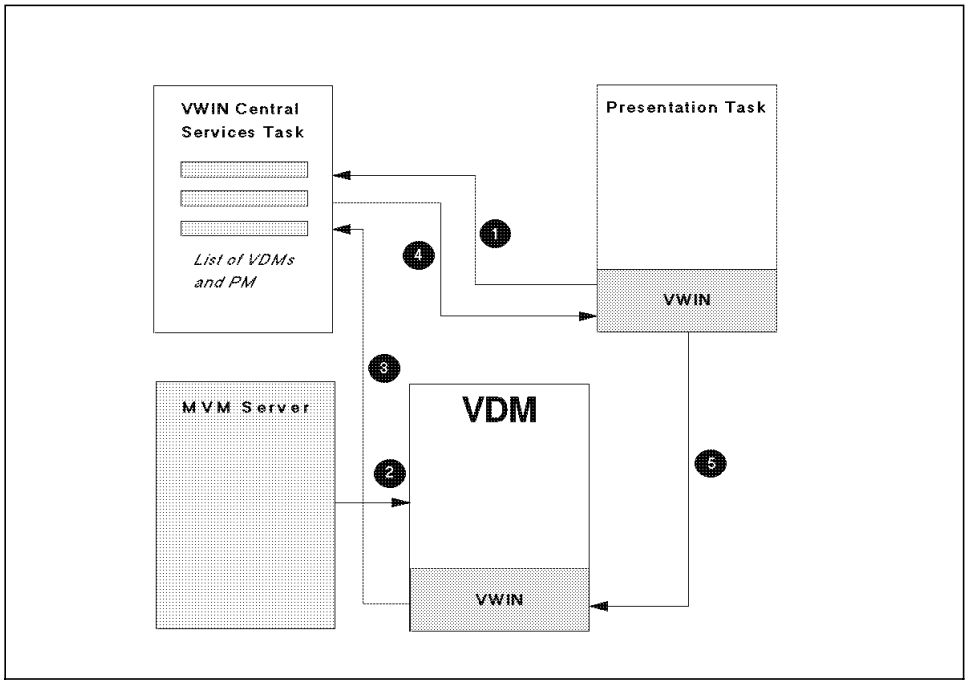


Figure 20. Overall Message Flow of OS/2 Warp Connect (PowerPC Edition)

1. When the presentation task VWIN initializes, it creates the VWIN Central Service Task. It provides it with a send right to the presentation task VWIN port.

2. The MVM Server creates the VDM. It passes to the new VDM a send right to VDM Central Services Task.

3. When the VDM VWIN initializes, it will send its information, along with a send right to itself, to the VWIN Central Services Task.

4. The VWIN Central Services Task then passes the VDM VWIN send right to the presentation task VWIN.

5. The presentation task VWIN then passes a send right to its port to the VDM VWIN.

In OS/2 Warp Connect (PowerPC Edition), the new video engine/video manager (VMAN) will provide the services found in the VWIN display driver routines. More discussion on VMAN could be seen in 4.3.3, “PM Video Device Driver” on page 92.

4.3.5 Fonts

This section will describe the font support of OS/2 Warp Connect (PowerPC Edition).

OS/2 Warp Connect (PowerPC Edition) will support generic fonts and can be used by all presentation drivers. The generic font can either be an outline or image font. The outline font technology supported depends on the Intelligent Font Interface (IFI) font drivers that are installed on OS/2 Warp Connect (PowerPC Edition).

The following figure shows font support for OS/2 Warp Connect (PowerPC Edition).

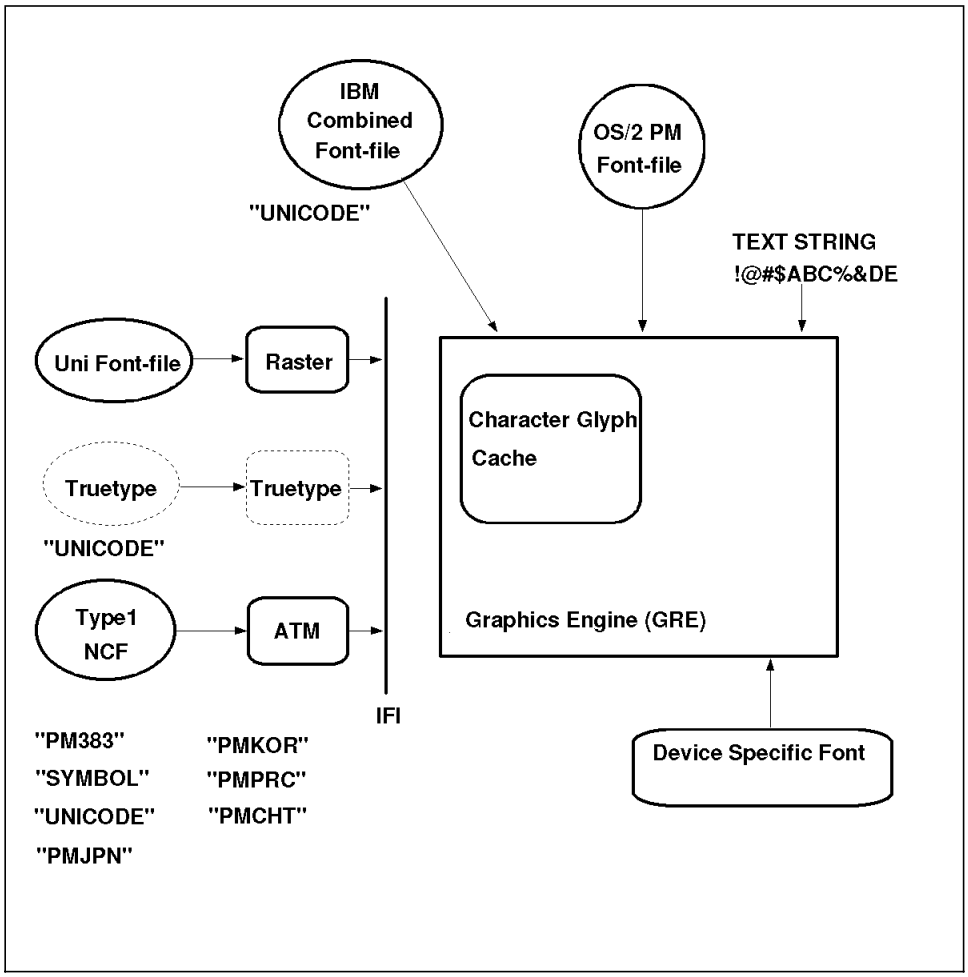


Figure 21. OS/2 Warp Connect (PowerPC Edition) Font Support

OS/2 Warp Connect (PowerPC edition) will support the following font formats as shown in Figure 21.

- IBM UNI font-file format
- IBM Combined font-file format
- OS/2 PM font-file format
- Adobe Type 1 font-file format
- Adobe Composite font-file format called NCF format
- Truetype

Glyph Fonts are applied to a table called a *code page*. A code page has several entries which contain different symbols. These symbols usually include letters, numbers, and special characters. Each of these symbols or images in a code page is called a *glyph*. Each entry in the code page is called a *code point*. A code point is an index into a code page to identify a glyph. For example if you take an ASCII code page, you would notice code point X′31′ would have a *glyph* for the character ″1″ defined in it.

Glyphlist The glyphlist name is the name that identifies a set of character glyph names and font index sequence of the character glyph set. All physical fonts are associated with a glyphlist name.

Glyph Index Translation
When an application draws a text string, the graphics engine parses the text string using a code page associated with the device context of the target device and translate it into font indices or indexes for the physical font. This is called *Glyph Index Translation*.

OS/2 Warp supports only the character glyphs which are defined in the PM383 glyphlist at present. Six more glyphlists have been added to OS/2 Warp Connect (PowerPC Edition). These added glyphlists enable the graphics engine to support new country and new language fonts.

OS/2 Warp Connect (PowerPC Edition) supports the following glyphlists:

- PM383

  383 Character glyphs are defined in this glyphs are defined in this glyphlist.

- UNICODE

  Consists of the UNICODE specification adopted by the International Organization for Standardization (ISO) as ISO 10646.

- SYMBOL

  Font specific encoding used to define a symbol character set.

- PMJPN

  PM383 extension for the Japanese language.

- PMCHT

  PM383 extension for the Traditional Chinese language.

- PMKOR

PM383 extension for the Korean language.

- PMPRC

  PM383 extension for the Simplified Chinese language

Graphics engine fonts: The graphics engine (GRE) reads and parses the following font formats directly:

- IBM Combined font-file format

  This a new format being supported by OS/2 Warp Connect (PowerPC Edition), which will be described later in this section.

- OS/2 PM font-file format

  This is the OS/2 2.1 font-file format which is designed for small character sets and will be supported.

ATM IFI font driver font: The ATM IFI font driver supports the following font file formats:

- Adobe NCF font-file format

  This is the Adobe new composite font format designed for small and large character set fonts.

- Adobe Type 1 font-file format

  This is the Adobe Type 1 font file format designed for small character set fonts. The Object Font Metrics (OFM) file format was enhanced in order to support the UNICODE glyplist.

Raster IFI font driver font: The following font-file format is supported by the Raster IFI font driver.

- IBM UNI font-file format

  The Uni font-file format is designed for image fonts which have large characters set in the font file such as the Double Byte Character Set (DBCS) and the UNICODE encoding. The Uni font-file format is a super set or extension format of the OS/2 2.1 PM font-file format.

Trutype IFI Font Driver Font: The Trutype IFI driver font is a new font driver that is developed to support the widely available true type fonts in the market place. The driver font supports the following font file format:

- Truetype font-file format

The following table is a summary of font format and glyplist support for OS/2 Warp Connect (PowerPC edition):

Table 4. OS/2 Warp Connect (PowerPC Edition) Font Format Support

Font-file formats	Glyphlists						
	PM383	UNICODE	SYMBOL	PMJPN	PMCHT	PMKOR	PMPRC
IBM Uni Font-file format	X	X	X	X	X	X	X
IBM Combined Font-file Format		X					
OS/2 PM Font-file format	X		X				
Adobe Type 1 Font-file format	X	X	X				
Adobe NCF Font-file format	X	X	X	X	X	X	X
Truetype	X						

Font Cache: The graphics engine implements character glyph (bitmap) caching for IFI font driver fonts.

The caching was designed to work for large characters set fonts with reasonable performance, hit ratio point of view. The following points were considered as the criteria for handling multiple large characters set font for the caching:

- High hit ratio for the given buffer size
- Linear search must be prohibited
- Fragmentation - glyphs have different sizes

Intelligent Font Interface (IFI): OS/2 Warp Connect (PowerPC Edition) graphics engine enhances the Intelligent Font Interface (IFI) in order to support raster IFI font drivers, additional glyphlists, etc. This IFI driver version will be defined as IFI version 2.1

Device Specific Font: A presentation device driver can provide device specific fonts for the built-in fonts in the hardware such as a printer or a display adapter card. For the device specific font the responsibility of the glyph index translation resides at the device driver. Device drivers must recognize which code page is associated with the text string and to the proper conversion to the font indices of the device specific font.

OS/2 Warp Connect (PowerPC Edition) Font object design: As we enter new markets with new character set requirements or attempt to support

multilingual text, we are facing the problem of building new UGLs or combining and using an existing one.

OS/2 Warp Connect (PowerPC Edition) uses a new model as shown in Figure 22 on page 114. An alternative approach was taken, this design supports font objects which maintain the information about the character sets they support. Each font object will have an associated DLL which maps the ASCII or UNICODE character encodings to the character glyph definition supplied by the font. This means that we remove the notion of the system maintaining a single list of supported characters, the UGL.

The advantage of this are:

1. The dependencies of the GRE on font encodings are isolated.
2. New character sets can easily be added through new fonts.
3. Multilingual text can now be supported through the UNICODE mapping support.

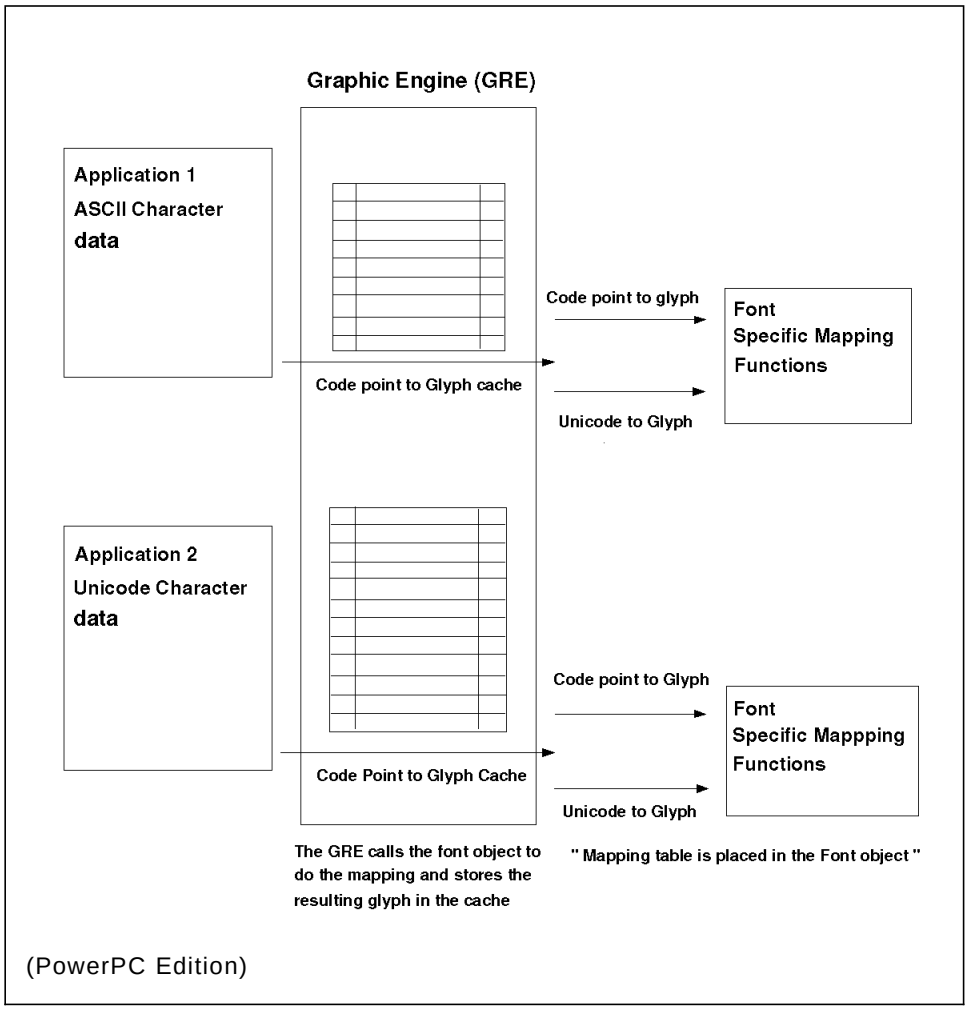


Figure 22. New Model for Font Architecture Support Under OS2 Warp Connect

The new model for OS/2 Warp Connect (PowerPC Edition) is shown in Figure 22. The glyphlist associated with the font is identified by the font header. This could be SYMBOL, UNICODE, PM383, PMJPN or some other name. Current bitmaps fonts have no header and are assumed to be PM383 bitmap fonts. The graphics engine (GRE) will use a table to associate the font with a DLL. The DLL will be associated with the glyphlist name.

This DLL supports three functions:

- Install
- CodePointToGlyph
- UnicodeToGlyph

The CodePointToGlyph and UnicodeToGlyph functions are used to do the mapping from either an ASCII codepoint or UNICODE to a 16 bit glyph index, which represents the appropriate character.

The font glyph index is used to query the intelligent font interface (IFI) driver of the font to retrieve the rasterized character bitmap. The install function is used to set up the mapping tables needed for the conversions and also establishes some of the font characteristics. Some of the values established at this point are:

- Number of glyphs in the font
- Lowest value of the font index
- Highest font index value
- Set of index ranges which contain the entire set of glyphs in the font

The Graphics Engine (GRE) deals with two kinds of fonts:

- **Engine Fonts**

  An engine font is a bitmap font with a defined structure. The engine fonts used today support only the characters in PM383. A table 383 entry is built with entry, size of the character and an 64k heap pointing to the character bitmap definition. This table is part of the interface to the Graphic Engine and is used by the graphic device drivers.

- **IFI Fonts**

  An IFI font is a font, outline or bitmap, and is accessed through a special interface called an IFI driver. The IFI driver accepts 16 bit glyph index and returns a rasterized image. The Graphics Engine creates a fake representation of an engine font for each IFI font that is loaded. As characters are rasterized, the entries in the header field are filled in. The display devices are not aware of the differences between engine fonts and IFI fonts. This mechanism is also a simple cache mechanism for small fonts. Large fonts use a secondary cache to store the rasterized bitmaps. Just before the display device is called to render the characters, the bitmaps are copied to the fake engine font format. In this way the display devices are also unaware of the difference between the

small and large fonts, however this will have a impact in performance for large fonts.

The engine font, *Font Transfer Area (FTA)* is supplied to the display drivers. This font will be used for existing support for PM display drivers.

The following algorithm is used as shown in Table 5. This is an example where *Unicode encoded* text and *ASCII encoded* text are used.

Table 5. Encoding Font Algorithm

No	Description
1.	Unicode or ASCII text is passed to the GRE to be rendered in a Presentation Space (PS).
2.	Identification of the current logical font for the PS will take place.
3.	If ASCII text is passed get the current PS codepage.
4.	Next the font's UnicodeToGlyph or CodePointToGlyph function will be called, which will return a font index for each character in the string.
	Either the native glyphlist of the font object is Unicode, which will result in a no operation, or
	The font object will convert from Unicode to the native encoding, that is PM383.
5.	Next the font cache will be searched for the bitmap.
	If it is a engine font then the bitmap will always be there, or
	If it is a IFI font, the bitmap may not be available.
6.	If the bitmap is not available, approach the IFI driver for the glyph, and place it in the cache.
7.	Copy the bitmaps to the Font Transfer Area (FTA), if required, this is not required if the font caching mechanism uses the FTA directly.
8.	The graphics driver is then called for the PS to render the bitmap.

This algorithm, together with the font objects, will support multiple glyphlists and Unicode in an efficient manner.

IBM Combined Font: Combined font originated from the limitation of OS/2 2.1 PM Universal Glyph List (UGL), which makes it difficult to support new languages and countries. OS/2 Warp Connect (PowerPC Edition) uses a *Combined Font* font architecture to overcome this limitation.

The following sections will show the design and concept and also the implementation of the combined font in the graphics engine.

IBM Combined Font A Graphics engine font that is dynamically created by combining several physical fonts. A application can select and use the font in the same way as a physical font.

A IBM Combined font consists of the following font entries:

- **Physical Font Entry**: A physical font entry is a Graphics Engine heap representing a Graphics Engine generic font. This contains information such as font metrics, glyphlist name. It uses count and pointers to retrieve character glyph data and character attributes.

  Physical font entries are created when a new font is installed through the GreLoadFont function, and will be destroyed when it is unloaded through the GreUnloadFont function.

  Physical font entries are stored in two different places in the Graphics Engine heap, which are in a global shared heap to be accessed by all processes, or in an instance heap to be accessed by only one process. The physical font entries for private fonts are placed in the instance heap, and the ones for public fonts are placed in a global shared heap.

- **Combined Font Entry:** Combined font entry is a binary representation of IBM Combined font-file format and is located in the Graphics Engine heap. The combined font entry includes font metrics for the combined font, an array of component font entries, use count and a priority list of component fonts. The component font must be a public font. The priority list of component fonts will be used to choose a component font from the UNICODE glyph index that is translated from the text string.

  Note

  The only supported glyphlist for combined font is UNICODE.

  Component fonts may have different glyphlist. In this case, the graphics engine will do the necessary conversion from UNICODE indices to indices of the appropriate glyphlist, in order to retrieve a character glyph from the component font.

  There are two passes in the process of the glyph index translation of a combined font. The first pass is to translate from code points to indices of the UNICODE glyphlist. The second pass is to translate the indices of the UNICODE glyphlist to the indices of the component font glyplist. The indices returned from the second pass will be used to extract font images and attributes from the component font file.

The combined font points to physical font entries in the graphics engine heap, the graphics engine must assure that all component fonts are useable for the combined font. Therefore, all physical fonts installed as public fonts must be loaded prior to loading the combined font.

A combined font entry will be created when an IBM combined font is installed through the GreLoadFont function, and will be destroyed when it is unloaded through the GreUnloadFont function.

Combined Font-File Creation Tool.

IBM provides a font creation tool for users and national language developers for creating a IBM combined font easily.

New Graphics Engine APIs

The following new APIs are implemented by the graphics engine.

- GreRealizeString
- GreQueryCharOutline
- GreQueryCharMetricsTable
- GreQueryCodePageObject

4.4 Graphics Subsystem Summary

From the user point of view, there are no major changes in Graphical User Interface. Users will see the same desktop and will use the same Workplace Shell as they did in OS/2 Warp.

The OS/2 Warp Connect (PowerPC Edition) Graphical Subsystem has been designed as a modular system to accommodate the fast growing hardware changes. This modular design enhances the ability for the developers to maintain and develop new code for future releases of OS/2 Warp Connect (PowerPC Edition).

The summary of Graphics Subsystem will be shown in Table 6 on page 119.

Table 6. A Summary of Graphics Subsystem

Item	Description
GRE	Some functions have been moved from Presentation Driver to Graphics Engine. It will be easy to maintain the Presentation Drivers.
	Graphics Engine will only map the device-independent output into specific device output and GRE2VMAN will act as a Device Driver Interface.
PMVDD	Uses a *GRADD* model that is modular in design to accommodate and assist driver developers to write drivers in stages.
Base Video Services	OS/2 Full-Screen is only a part of the PM windowed application without the border. OS/2 sessions are swapable between windowed and full screen.
	OS/2 Warp Connect (PowerPC Edition) will not use Base Video Handler. All the functionality has been moved to OS2CHAR and VIDEOPMI.
Fonts	New font-file formats have been added with more character glyphs support for these new formats.
	Uses a combined font architecture whereby new languages and countries can be added easily.

4.5 Printing Services

Printing Services in the OS/2 Warp Connect (PowerPC Edition) operating system provide functions equivalent to those that are currently available in OS/2 Warp (Intel). The requirements that the printer server (spooler) had to meet to provide this support include:

- Allow printing from both OS/2 and MVM (DOS).
- Print using the same interfaces as those used for display.
- Print data which is already in the print stream.
- Allow for serialization and prioritization of print jobs.
- Provide an application interface to allow attributes to be specified for print jobs.
- Allow printing from current OS/2 and Windows applications.

The most obvious change in the printing system when comparing it to OS/2 Warp (Intel) is that, as with the other components of the architecture, the IBM

microkernel is used when accessing hardware devices. Figure 23 on page 120, shows the utilization of the IBM microkernel when an application is printing.

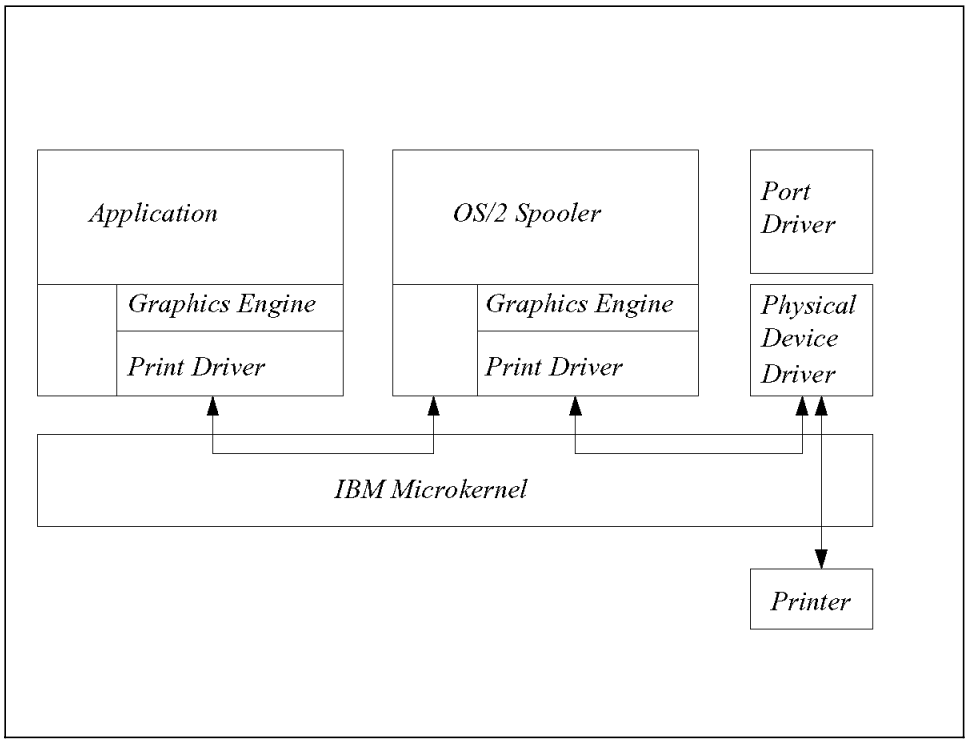


Figure 23. Printing in OS/2 Warp Connect (PowerPC Edition)

The components of the printer server are discussed is the following sections.

4.5.1 Spooler Objects

The OS/2 Warp Connect (PowerPC Edition) spooler function is provided to the system by a number of objects. Mostly these objects have been moved across from the OS/2 Warp (Intel). The spooler objects available in OS/2 Warp Connect (PowerPC Edition) are outlined below.

4.5.1.1 Logical Printer (Queue)

The logical printer matches the current OS/2 print queue. The user may have as many of these as they desire.

There may be multiple Print Drivers and Port Drivers associated with a Logical Printer. However, at the time a spool job is opened, one of them must be selected. Normally, the default Print Driver and Port Driver are used.

4.5.1.2 Print Driver

The print driver matches the current OS/2 print driver object. The print driver determines the Printer Description Language used (for example, Postscript, PCL5, Dot Matrix).

The print driver has a set of attributes. These are specified on a per printer, or per job basis (print properties or job properties). Many attributes can be specified in either place.

4.5.1.3 Port Driver

The port driver is a high level controller for data going across a port. It provides a high level interface to the physical device. The port driver handles bi-directional interface with the printer and maintains the current state of the printer.

In OS/2 Warp (Intel), the port drivers were significantly enhanced to support the needs of bi-directional communications. The function of the port driver has been implemented in OS/2 Warp Connect (PowerPC Edition). However, bi-directional printer communications will not work since the required device driver support is not yet available.

The persistent state of the port is kept as a set of attributes to the port driver in the registry. Normally, this data is only accessed by the spooler.

4.5.1.4 Spool Job

The spool job is created when an open is done of the logical printer. The spool job inherits the attributes from the logical printer, print driver and port driver. Many of these attributes may be specified on an individual job basis.

4.5.2 Printing from DOS and Windows

There are several methods of printing from DOS and Windows:

- Use the Windows Print interfaces which print using the GDI APIs using the same method as is used for display.
- From DOS, copy a file to a port using the copy of print command, or an application which opens the port as a file. DOS selects the logical printer associated with that port.

  DOS printing is done by have the OS/2 spooler perform a DosFSAttach in place of the parallel port for logical devices supported by the spooler (LPTx).

  The MVM Server provides a virtual device driver (VLPT) which handles the DOS side of this communication.

4.5.3 Printer Driver Support

OS/2 Warp Connect (PowerPC Edition) provides support for the following printers. All of the drivers are 32 bit C presentation drivers which are source compatible between OS/2 Warp (Intel) and OS/2 Warp Connect (PowerPC Edition). Unlike the current IBM printer drivers which are already 32 bit C drivers, most non-IBM print drivers will not be easily recompiled because they are written in 16 bit assembler. However, this list includes almost all of the following PC attached printers:

- IBMNULL (passthru)
- PostScript (level 1, level 2)
- HP LaserJet (III, 4)
- PPDS (4019, 4029)
- Omni driver (HP DeskJet, HP PaintJet, Epson)

4.6 System Management.

One of the goals for OS/2 Warp Connect (PowerPC Edition) is to learn from past experiences and create a product that can be supported in the field in a timely and cost effective manner.

In OS/2 Warp (Intel) kernel level debugging was used to isolate the failures that were encountered in the system. The OS/2 Warp Connect (PowerPC Edition) design depends instead on the serviceability-oriented instrumentation as the primary means of debugging problems. This is a

radical departure from the traditional debug techniques, and has resulted in a system that can be managed in a more intelligent fashion.

Additionally, since much of the systems management architecture is intended to be shared with OS/2 Warp (Intel), the serviceability of the OS/2 Warp Connect (PowerPC Edition) enhancements will eventually be incorporated within the OS/2 Warp (Intel) platform.

4.6.1 Installation

As mentioned in section 5.2, “Feature Install” on page 131, the OS/2 Warp Connect (PowerPC Edition) install process treats all system components as features. Based on this philosophy, the system management component is installed as a base feature.

The system management install process installs all of the system management components and consists of 3 parts:

- Module Install
- Configuration Information
- Workstation identification information

The installation, by default, places the systems management components into the OS2SYSTEMRAS directory of the boot drive. Once completed, the following ICONS will be available to the end user from the Systems Management folder:

- **Syslog Display** - Utility to display error log details.
- **FFST Configurator** - First Failure Support Technology (FFST) configuration utility.
- **Dump Formatter** - PMDF Dump formatter.
- **Trace Formatter** - Utility to display trace entries.
- **SYSLEVEL** - Utility to query system configuration.
- **System Dump Configurator** - System Dump configuration.
- **DMI MIF Browser** - Utility to display Vital Product Data (VPD) information.

4.6.2 System Management Initialization Process

In order to obtain the necessary information or data for systems management, OS/2 Warp Connect (PowerPC Edition) contains a system management initialization process called SMSTART. The SMSTART module is invoked by the OS/2 Server during startup of the OS/2 Warp Connect (PowerPC Edition) operating system.

SMSTART is a simple program that provides the following services to the system:

- Starts a thread that handles the enablement part of the system dump.
- Creates a thread for each of the following processes and starts them in the following order:
  - First Failure Support Technology (FFST) microkernel service daemon
  - Desktop Management Interface (DMI)
  - Error log daemon
  - Trace daemon
  - FFST daemon

These process are continuously monitored and restarted if they die.

If the SMSTART process experiences problems in starting one of the processes, other than the error log daemon, then it will append the reason to the error log. If SMSTART is unable to start the error log daemon, or is having problems starting its own process, then it will log the error to an internal file (SMSTERR.DAT) on the boot drive, and in the OS2SYSTEMRAS directory.

4.6.3 Serviceability Tools

In OS/2 Warp Connect (PowerPC Edition), the serviceability tools are programs that have been supplied with the system in order to provide system management capabilities. The serviceability tools belong to several different classes, each of which provides a different level of system management information to the user.

The serviceability tools architecture is layered to provide a hierarchy of tools:

- First Failure Data Capture (FFDC)
  - Error Logging
  - First Failure Support Technology (FFST)

- Data Browsing
  - Event Tracing
  - Resource Monitoring
  - Performance Monitoring
- System Dump
- Low Level Remote Debug

Information on how to use the various tools described in this section is found in the on line book *Systems Management User's Guide*, in the Systems Management folder.

4.6.3.1 First Failure Data Capture

First Failure Data Capture (FFDC) is intended to decrease the need for the reproduction of user failures by automatically capturing the data associated with a failure as soon as it is detected. In order to accomplish this, the software modules in OS/2 Warp Connect (PowerPC Edition) have been coded in a defensive manner which allows them to detect aberrant conditions. Once such a failure is detected, key internal failure states will be logged for subsequent analysis. The primary FFDC tool is the First Failure Support Technology (FFST) which utilizes the logging service.

All system logs, including the all important central error log, are created and stored in the OS2SYSTEMRAS directory of the boot drive.

4.6.3.2 Data Browsing

Data browsing allows service personnel to query and sample running systems in order to better understand the operation of those systems. Data browsing is the least invasive from of serviceability tool. It should not affect the operation of a running system. The primary data browsing tools in OS/2 Warp Connect (PowerPC Edition) support Event Tracing, Resource Monitoring and Performance Monitoring.

Event Tracing: Event tracing in OS/2 Warp Connect (PowerPC Edition) has been designed to incorporate the best aspects of the currently available tracing methodologies. The trace facility has been built upon the base Event Trace capabilities of the microkernel which allows the facility to be used by the microkernel, by the microkernel services and by OS/2 applications.

The event tracing facility is comprised of a number of tools. They include:

- TRACE - This utility is used to turn on and off trace points.

- TRACEFMT - This utility is used to view logged event trace data.
- TRCUST - This utility is a trace point definition tool.

The event trace facility has been built on the existing OS/2 trace facility. It has been expanded to include support for the new OS/2 Warp Connect (PowerPC Edition) components (for example, the microkernel). The changes will be incorporated into the OS/2 Warp (Intel) version of the operating system.

Performance Monitoring: The main tool provided in OS/2 Warp Connect (PowerPC Edition) for performance monitoring is called the System Performance Display program (SPD) for OS/2 Warp Connect (PowerPC Edition). It is designed to assist end users in analyzing system performance. The SPD helps in managing the major system resources (CPU, DASD, paging and memory) to achieve greater efficiency and maximum performance.

The SPD tool allows for collecting of performance data about the operating system and applications, displaying this information graphically, and creating reports or providing statistics on the collected data.

Resource Monitoring: Resource monitoring allows the user to obtain data on usage of the various system components. Since this process is tightly coupled with performance monitoring, the Systems Performance Display program contains features that allow resource monitoring to occur. For example, the SPD program, has a memory analysis tool which allows memory analysis at various levels, including the working set.

4.6.3.3 System Dump

It is possible that First Failure Data Capture (FFDC) will not be sufficient to solve all detected system and application problems. In some cases it will be necessary to use the broader approach of taking a full system dump.

OS/2 Warp Connect (PowerPC Edition) supports an automatic system dump capability that can dump system images to a defined disk area. The system dump mechanism also has the ability to automatically reboot the system.

The OS/2 Warp Connect (PowerPC Edition) system dump process consists of three parts:

- Configuration/Enablement

  The covers all the preparatory activities that occur before a system dump is taken. Control of this function is through the System Dump Configurator program.

- Triggering

  As with OS/2 Warp (Intel), the system dump can be triggered by keyboard entry sequences, programmable APIs or CONFIG.SYS entries. One difference is that the system dump is now taken by the microkernel.

- Formatting

  OS/2 Warp Connect (PowerPC Edition) provides a system dump formatter to display all portions of the system dump.

4.6.3.4 Low Level Remote Debug

When occasions occur that FFDC and System dumps do not provide enough information, OS/2 Warp Connect (PowerPC Edition) also has the ability for remote debug.

The remote debugging facility is based on a core kernel-level debugger that is included in all OS/2 Warp Connect (PowerPC Edition) systems (as part of the microkernel product).

The debug system can be accessed by attaching another machine to the serial port of the PowerPC machine and using the debugger through a communications program.

4.6.4 Vital Product Data

The Vital Product Data (VPD) facility is a new capability for the OS/2 environment. It provides a standard way in which to describe the hardware and software parts of a system. VPD information is used for a variety of purposes that include:

- Enables the creation of serviceability tools that display what components are present of a customer′s system.
- Allows standard component identification information to be included on all logged error records.
- Enables the installation tools to determine the current state of product prerequisites and co-requisites on a customer′s system.
- General system management.

The VPD facility has been based on the Desktop Management Interface (DMI) standard, which was developed by the Desktop Management Task Force (DMTF).

4.6.4.1 The DMI Standard

The DMTF DMI standard is an attempt to create a local operating system framework that links network management agents to the hardware and software components running on the system.

The DMI standard addresses four levels of definition:

- An API set (called the Management Interfaces (MI)) that can be called by management applications (or their agents).
- A Logging Service Interface (SPI) set (called the Component Interface (CI)) that allows hardware and software components to respond to MI requests.
- A file-based syntax (called the Management Information Format (MIF) syntax) that allows DMI-compliant components to be defined to the DMI Service Layer (SL).
- A set of standardized group definitions that define classes of attribute sets that are shared by classes of components.

The heart of the DMI architecture is the MIF database. As DMI components are installed, their MIF file definitions are installed within the MIF Database. As with a products relating to systems management in OS/2 Warp Connect (PowerPC Edition), the MIF database resides in the OS2SYSTEMRAS directory on the boot drive.

In OS/2 Warp Connect (PowerPC Edition), access to the VPD information is through the DMI MIF Browser program.

Chapter 5. Installation

The OS/2 Warp Connect (PowerPC Edition) installation process has changed significantly from the current OS/2 Warp (Intel) version of OS/2. The first release of the installation program provides the ability to install the operating system, its features, device drivers, and other applications. Applications can be pure OS/2 applications, OS/2 applications with system service components, or pure system services that run on the IBM microkernel.

In this release of OS/2 Warp Connect (PowerPC Edition), the Configuration Installation and Distribution (CID) methodology of installing of the operating system is not possible. However, CID installation is still available for the installing of CID enabled applications into the OS/2 Warp Connect (PowerPC Edition) environment.

The OS/2 Warp Connect (PowerPC Edition) installation is described in two phases. The goal of the first phase, called media preparation, is to prepare the machine for the installation of the OS/2 Warp Connect (PowerPC Edition) system. The second phase, feature install, allows the installation of the OS/2 Server and optional features.

5.1 Media Preparation

The media preparation phase of the OS/2 Warp Connect (PowerPC Edition) installation is similar to the initial phase of the OS/2 Warp (Intel) version of the operating system. The aim of this phase is to prepare the machine for the installation of the operating system by partitioning the hard disk into the necessary configuration.

During this phase of the installation, the following sequence of events occurs:

1. Initial boot from the CD-ROM installation media.
2. Load and start the Microkernel, Microkernel Services, Shared Services, and the OS/2 installation program from the installation media.
3. Load and start the media preparation program by the OS/2 initialization program.
4. Partition the hard disk using the utility file server (FDISK).
5. Create the initial file system(s) using the utility file server (FORMAT).
6. Return to the OS/2 initialization program.

7. Load and start the OS/2 Server from the installation media with PM Shell activated.
8. Copy files of MK and OS/2.

The IBM Microkernel relies on device configuration data, and resource description, to properly configure a device. Depending on the system architecture, the device configuration information is obtained from a variety of sources.

5.1.1 Partitioning

One of the main differences in OS/2 Warp Connect (PowerPC Edition) and OS/2 Warp (Intel) is the requirements for the partitioning of the hard disk. Instead of a single partition that was used in OS/2 Warp (Intel), two partitions are created OS/2 Warp Connect (PowerPC Edition). A typical example of an OS/2 Warp Connect (PowerPC Edition) disk configuration is shown in Figure 24. The two partitions are known as the:

- Boot or Type 41 Partition
- System Partition

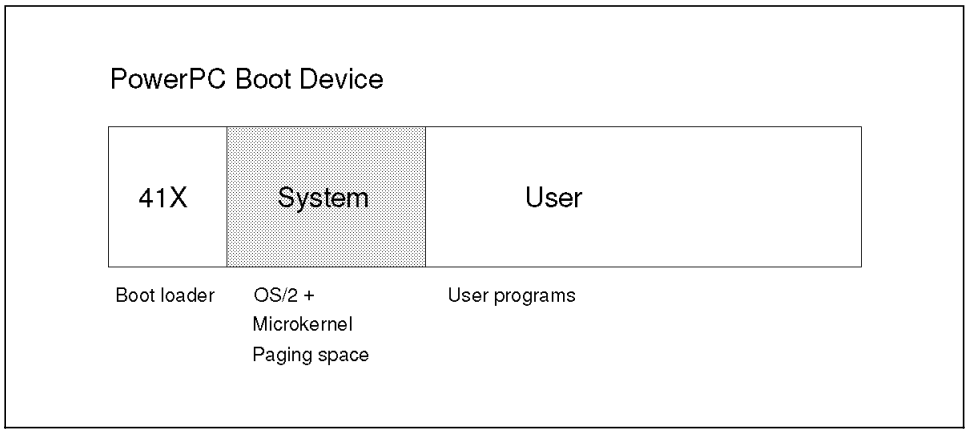


Figure 24. Disk Partitions in the Boot Device

The boot partition contains the boot loader. The boot loader is a requirement by the firmware of the PowerPC and is loaded by the firmware when the system boots. The boot partition is a small partition of approximately 1-2MB.

The system partition is the main partition of OS/2 Warp Connect (PowerPC Edition). It is entered into the name space as (/file/system) and contains the following OS/2 Warp Connect (PowerPC Edition) components:

- The Microkernel, Microkernel Services, System Services, Device Drivers and control files necessary to correctly execute the microkernel.
- The OS/2 Server and its associate support files.
- Enough space to support paging during the boot process. Once the system has started, the OS/2 Server will start an additional paging system for its own needs.

In the first release of the operating system, the system partition is a primary partition, formatted using the FAT file system. There is no option during the installation to choose the file system for the system partition. HPFS formatted system partitions may be an option in a future release of the operating system.HPFS partion is supported in the user partition.

5.1.2 System Migration

OS/2 Warp Connect (PowerPC Edition) can be installed over any previous PowerPC based operating system. However, there are a number of restrictions in what information can be migrated to the new installation.

If you are re-installing the system over a previous copy of the OS/2 Warp Connect (PowerPC Edition) operating system, then the system will migrate exiting applications and programs.

If you are installing over one of the two currently available PowerPC operating systems, Windows NT (PowerPC Edition) or AIX, then no system migration occurs. In fact if the native file system of Windows NT or AIX is being used (for example the NTFS file system for Windows NT) then the hard disk must be reformatted during the first phase of the OS/2 Warp Connect (PowerPC Edition) installation.

In OS/2 Warp (Intel) the boot manager facility allowed different operating systems to co-exist on the same hard disk. In this release of OS/2 Warp Connect (PowerPC Edition), a boot manager facility has not been provided. A multi-boot facility which would allow a combination of PowerPC based operating system may be available in a future release.

5.2 Feature Install

Feature install handles the installation of documentation, games and the BonusPak.

Feature install also handles maintenance of software in that same way that it handles new software installation. This means that unlike in OS/2 Warp (Intel) a separate service installation tool is not necessary.

Although the new installation routines will allow for cumulative and selective fixes to be applied to the system, for the first release of the OS/2 Warp Connect (PowerPC Edition) operating system, backout of service installations is not supported.

5.2.1 Feature Install Catalog

The first screen that is shown during the installation is the Feature Install catalog. Not unlike the selective install program in OS/2 Warp (Intel), the Feature Install program allows you to choose to install documentation, games and BonusPak.

5.2.2 Drag and Drop Install

One of the most innovative features of the feature installation program is that each feature that is installed is considered an object by the system. This is totally different to the OS/2 Warp (Intel) installation program and allows a level of configurability that was unavailable in previous OS/2 versions. In order to accomplish this, the feature installation program adds two objects to the Workplace Shell environment. They are:

- Install Object
- Inventory Object

Both of these objects have settings that can be changed by the user. The install objects can be opened to make selections and configuration changes, while the inventory objects contain information about system software installed in the system.

5.2.3 Install Objects

The install object represents a product that can be installed to your system. The object exists on the product source media, and can be accessed through the file system object.

For example, a product that is shipped on CD-ROM could be represented by an install object in the root directory of the CD-ROM. In this case, the user could view the product's install object by placing the CD-ROM in the drive and selecting **Open - icon view** from the context menu of the CD-ROM drive object. The install object would be visible alongside any file objects in the top container of the opened CD-ROM object.

For OS/2 Warp Connect (PowerPC Edition), the existence of the install object allows you to bypass the Feature Install catalog and install the various components directly from the install object.

The install object class supports commonly used installation functions (such as file copy, configuration file updating, and product registration).

Once the install object is visible, you may install all defaults for the product by simply dragging the install object to any Workplace Shell folder, including the desktop itself. Alternatively, you could perform actions on the install object to configure the product before installation.

From the install object context menu, the user can open a tree view of the object (to obtain access to a lower level of install objects) or open the object settings (to view product information).

5.2.3.1 Available Modes

Install objects support two different modes of operation. They are:

- Development
- User

Development mode enables the creation of install objects. This mode is used by software developers. User mode allows for the installation of install objects. This mode is used by the end user. For example, the end user should not be able to modify the information on the install object settings pages.

5.2.4 Inventory Objects

An inventory object represents a product that has been installed to the your system. This object may be used to remove the product the product from the system (uninstall), or simply to view the features and settings that were selected when the product was installed.

One advantage of the inventory objects is for CID administrators. Each inventory object provided by the Feature Installer stores its persistent data in an associated text file called INSTDATA.INI. This file contains the CID keywords that were used to create the object. This means that creating a CID response file merely involved copying the information from the INSTDATA.INI files to a new response file.

5.2.4.1 Views

The inventory hierarchy parallels the hierarchy of install objects that were selected when the product was installed.

5.2.4.2 Removing Features

The inventory objects offer greater control over removing operating system features than was available in OS/2 Warp (Intel). In order to remove a feature from the system, simply drag the inventory object representing the installed product to the shredder, or by selecting uninstall from the open view or from the objects context menu.

This will result in the deletion of the product files, the reversal of any configuration changes made when the product was originally installed, the deletion of any workplace objects created for the product, and the deletion on the products inventory object.

The Inventory objects are kept in an Installed Software folder. The Installed Software folder is available from the OS/2 desktop.

Although this system is somewhat more powerful than the uninstall feature of OS/2 Warp (Intel), it does not support the removal or the entire operating system. If you wish to install a different operating system onto the machine, then the most common practice is still to use FDISK and FORMAT, in the same way that OS/2 Warp (Intel) is working currently.

5.2.4.3 Adding Features to the System

Adding additional features once a component is installed is very easy. To add features or components that were not originally selected simply follow one of the steps outlined below:

- Re-open the install object and drag the desired subfeature to the corresponding inventory object.
- Select install from the context menu for the feature.

5.3 Inventory Information

Documentation of what is installed on the system, and the level of the software on the system is maintained in an installed software inventory. This ensures that service applied to the system will not return the user to a previous level of the software. Using the installed software inventory, also provides service with an accurate description of products and services installed onto a machine. This makes it easier for service to give a customer

a response to his problem, and it gives service the ability to more accurately duplicate a customer′s system for problem re-creation.

The following information is recorded in an installed software inventory about each product subproduct installed:

- **Description** - A description of the software.
- **Tag** - A short name for the software.
- **Title** - The software package title.
- **VendorTag** - A short name for the software manufacturer.
- **VendorTitle** - The name of the software manufacturer.

5.4 CID and Unattended Installation Support

All of the installation programs and mechanisms in OS/2 Warp Connect (PowerPC Edition) support the Configuration, Installation, and Distribution (CID) architecture which calls for support of installation from redirected sources and installation in an unattended fashion. This entails:

- Redirected installation
- Response file support
- Ability to transfer product files / images to a code servers hard disk for installation purposes
- Command line support is implemented via Clifi executable
- Logged process information

In this release of the OS/2 Warp Connect (PowerPC Edition) operating system, CID support differs slightly from what is available in OS/2 Warp (Intel). Although it is still possible to install applications and fixes using the CID methodology, the operating system itself cannot be installed using CID.

5.4.1 Standard Keywords

The following standard keywords are supported by the OS/2 Warp Connect (PowerPC Edition) Feature Installer:

- **Include** - Include other response files for processing
- **Reinstall** - Force reinstallation, even if the product is up to date (Implemented only through Clifi).
- **Append** - Append log information to log files (Implemented only through the Clifi).

Several of the commonly used keywords are no longer supported by the OS/2 Warp Connect (PowerPC Edition) Feature Installation program, and are shown in Table 7 on page 136.

Table 7 (Page 1 of 2). Unsupported CID Keywords in OS/2 Warp Connect (PowerPC Edition)

Keyword	Definition	Comment
Copy	Copy a file.	This keyword will be ignored by the OS/2 Warp Connect (PowerPC Edition) Feature Installer
UserExit	Execute a user exit program.	This keyword will be ignored by the OS/2 Warp Connect (PowerPC Edition) Feature Installer. The Installer provides a rich ability for a package developer to specify user exits.
Reconfigure	Force product reconfiguration.	The OS/2 Warp Connect (PowerPC Edition) Installer does not separate the file transfer and configuration steps. Configuration will always be done.
Software	Identify product features.	This keyword will be ignored by the OS/2 Warp Connect (PowerPC Edition) Feature Installer. The Installer uses a *SELECTED* variable for each object that is to be installed. The value of this variable can be changed by the actions that occur as a result of processing dependencies that the package developer specified. Within the appropriate sections of the response file the *SELECTED* keyword my be used to selected the installation of specific software. Dependency processing will override any setting specified in the response file.
Defer_Configure	Defer product configuration.	This keyword will be ignored by the OS/2 Warp Connect (PowerPC Edition) Feature Installer.

Table 7 (Page 2 of 2). Unsupported CID Keywords in OS/2 Warp Connect (PowerPC Edition)

Keyword	Definition	Comment
Icon_Placement	Specify positioning of product icons within a folder.	This keyword will not be produced or directly processed by the OS/2 Warp Connect (PowerPC Edition) Installer. However, it will be parsed and made available to product developers within user exists. If included in a response file, it needs to be placed in the appropriate section. When name qualification will not remove an ambiguity, only the last one encountered will be remembered. This function is handled via the Create Objects pages through the setup string parameter. Variables can be used within the setup string to control the icon placement.
Folder_placement	Specify positioning of product folders.	This keyword will not be produced or directly processed by the OS/2 Warp Connect (PowerPC Edition) installer. However, it will be parsed and made available to product developers within user exists. If included in a response file, it needs to be placed in the appropriate section. When name qualification will not remove an ambiguity, only the last one encountered will be remembered. This function is handled via the Create Objects pages through the setup string parameter. Variables can be used within the setup string to control the folder placement.

5.5 Tracing Installation Problems

One of the most common problems with OS/2 Warp (Intel) was with the installation of the product. Users were often left with a system that has failed to install, but without any indication to what portion of the installation failed.

In order to rectify at least some of this problem, OS/2 Warp Connect (PowerPC Edition) supports the use of First Failure Support Technology (FFST) for error logging during the installation. It also uses FFST for its data browsing capabilities to externalize the relevant state of the component, both during system boot from a CD-ROM and during selective installation of subsystems. Externalizing the state of a component allows the user to examine what action the component was performing when it may have failed.

FFST error logging is always enabled. The default error log is kept in the install target partition during a CD-ROM boot, unless otherwise specified in the response file. In this case, the response file would be located on diskette.

In addition to the FFST support, the OS/2 Warp Connect (PowerPC Edition) Install component keeps a user-defined error and activity log independent of FFST in order to meet CID requirements.

5.5.1 Media Preparation

Only a limited number of error messages can be logged using First Failure Support Technology (FFST) before the OS/2 server is running. During the media preparation, OS/2 Warp Connect (PowerPC Edition) uses one FFST call to indicate an error has occurred and direct the user to the more detailed install error log, called INSTALL.LOG, which is kept in the target partition.

Entries will be recorded in the install log for the following:

- **Error Messages** - For errors detected during media preparation and the formatting of partitions
- **Major System Changes** - Such as disk partitioning and formatting of partitions

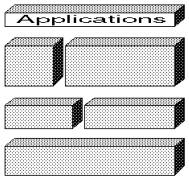
5.5.2 Feature Install

Due to the object oriented and graphical nature of the Feature Installation component, much of the internal state of the installation has already been externalized. For example, feature selection and dependencies, variable resolution, and file lists are already accessible to the user and support personnel.

The OS/2 Warp Connect (PowerPC Edition) Feature Installation component provides the following information through FFST:

1. Procedure tracepoints. Major functions are traced to provide sufficient information to understand component failures.
    a. File transfer functions
        1) Source and target filenames, including path
    b. Object creation and class registration
        1) Setup string used to create the object
        2) Setup string used to register the class
    c. Configuration changes to CONFIG.SYS, OS/2 and Windows INI files
        1) Lines deleted
        2) Lines added
        3) Lines modified
    d. User Exit APIs
        1) Feature ID
        2) Function called
2. Error Messages. Error messages detailing component failure are logged with FFST, as well as with a user-selected log file for CID purposes.
    a. Configuration change errors
    b. File transfer
    c. Internal install functions
    d. Object creation

Chapter 6. Application Support



OS/2 Warp Connect (PowerPC Edition) runs 32 bit OS/2 applications, and DOS, Windows or DPMI applications. It is also able to run Win32s applications as OS/2 Warp (Intel) does. The 32 bit OS/2 applications, if currently available on OS/2 Warp (Intel), need to be recompiled for the PowerPC platform.

The DOS, Windows or DPMI applications run unchanged from the OS/2 Warp (Intel) environment. OS/2 Warp Connect (PowerPC Edition) provides the necessary emulation of Intel instructions to run the DOS, Windows or DPMI binaries.

OS/2 Warp Connect (PowerPC Edition) will not run 16 bit OS/2 applications, nor will it run family API (FAPI) applications.

6.1 Application Development

Currently, application development is done by the means of the Metaware cross platform development tools. This means that the compile and link of a program is done on the Intel platform, and that the executable program or dynamic library produced, has to be run on the PowerPC platform. The main components of the Metaware toolkit are:

- HCOPPC - The Compiler

  The compiler may be directed to invoke the other necessary components of the toolkit. If so, it creates PowerPC assembler statements and invokes the PowerPC assembler.

  The compiler must be on, or beyond Release 2.6.

- ASPPC - PowerPC Assembler

  The assembler creates an ELF compatible object file from either a source file created by the compiler or a source file written by a developer.

  The assembler must be on, or beyond Release 1.74.

- LDPPC - The Linker

  The linker may be invoked by the compiler, or it may be run separately. In both cases it links any ELF compatible object modules, regardless of their source origin, and creates an ELF executable or dynamic link library.

  The linker must be on, or beyond Release 3.47.

- ARPPC - The ELF Archiver

  The archiver manages library files by combining .OBJ files or .LIB files into new .LIB files, deletes entries in .LIB files and lists entries in .LIB files.

- ELFDUMP - The ELF Dumper

  The ELF dumper knows the ELF format, and presents the contents of an ELF compliant file in readable form.

- BD - The Binary Dumper

  The binary dumper is a hexadecimal browser, which might be instructed to recognize certain data structures.

The toolkit also contains documentation in the form of postscript files. It contains the necessary header files, both the standard C and C++ header files, and the OS/2 header files normally found in the OS/2 toolkit.

The .LIB files associated with the above mentioned header files are present, as well as some sample programs. Finally, Direct-to-Som is supported in the package.

Appendix A. Changes to MVM DOS Settings

Table 8 (Page 1 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
COM_DIRECT_ACCESS	In OS/2 Warp Connect (PowerPC Edition), direct access to hardware applications is not permitted by the hardware. Consequently, OS/2 Warp Connect (PowerPC Edition) will *always* emulate application access of hardware ports.		√	
DOS_BUFFERS	Comparable to the BUFFERS= statement in PCDOS, which is used to allocate memory for a specified number of disk cache buffers when the system starts.	√		
DOS_CODEPAGE	This setting allows the user to specify the codepage of a DOS session. The codepage must be already available in the system, allowing the user to choose from an available list.	√		
DOS_COUNTRY	This setting allows the user to specify the country code for a DOS session. The country code must be already available in the system, allowing the user to choose from an available list.	√		

Table 8 (Page 2 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
DOS_FCBS	This DOS settings is carried over from OS/2 Warp (Intel), but the defaults have changed from those used by OS/2 Warp (Intel). This setting is used to specify the maximum number of FCBs (file control blocks) that can be open concurrently in one DOS session. One of the two possible parameters on this setting has been removed; the default has been changed from 16 to 4.			√
DOS_FCBS_KEEP	Under OS/2 and versions of DOS prior to 6.3, there are two FCB values. The second value is no longer utilized, meaning that this entry is no longer required.		√	
DOS_INSTALL	Loads a memory-resident program into memory when you start a DOS session. The memory-resident programs stay in memory as long as your sessions exist, and can be used even when other programs are active.	√		
DOS_MESSAGE_FILE	This setting specifies the language that will be used for a session.	√		
DOS_NUMLOCK	This setting specifies whether the NUM Lock key on the keyboard is set to ON or OFF when the VDM starts.	√		

Table 8 (Page 3 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
DOS_STACKS	This setting supports the dynamic use of stacks to handle hardware interrupts. STACKS=n,s. n specifies the number of stacks. Valid values for n are 0 and numbers in the range 8 through 64. s specifies the size (in bytes) of each stack. Valid values for s are 0 and numbers in the range 32 through 512. The default for this setting is 9,128.	√		
DOS_SWITCHES	This setting provides special options, useful only from the config.sys file. The options are: • /K, forces an enhanced keyboard to behave like a conventional keyboard. • /N, prevents you from using the F5 or F8 key to bypass startup commands. • /F, skips the delay after displaying the 'Starting PC DOS...' message during startup.	√		
DPMI_DOS_API	The default for this setting has changed from AUTO to ENABLED.			√
EMS_LOW_OS_MAP_REGION	This setting was designed specifically to support Microsoft Windows Version 2.0 real mode applications. OS/2 Warp Connect (PowerPC Edition) will not support Windows 2.0 applications. Hence, this setting is no longer required.		√	

Table 8 (Page 4 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
GENERIC_HW_SUPPORT	This setting allows the user to specify if they want support for generic devices or not.	√		
HW_ROM_TO_RAM				√
IDLE_MAX_SLEEP_TIME	This setting defines the maximum period of time, in milliseconds, that the DOS session will be put to sleep when it is determined that the DOS session is idle. A value of 0 will disable the idle detection in this DOS session. The default setting is 2000 milliseconds.	√		
IDLE_SENSITIVITY	In OS/2 Warp (Intel), this setting was used to set a threshold for polling time before the operating system reduced the polling programs portion of the processor time. This setting has been replaced by: • IDLE_TIMEOUT • IDLE_MAX_SLEEP_TIME		√	
IDLE_TIMEOUT	This setting defines the amount of time (in seconds) allowable between the last busy event and an idle event. A value of 0 will disable idle detection in this DOS session. The default value is 5 seconds.	√		
TRANSLATED_CACHE_SIZE	This setting allows the user to specify the size of the Intel translated instruction cache.	√		

Table 8 (Page 5 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
VIDEO_FASTPASTE	This setting exists in OS/2 Warp (Intel). Its purpose is to allow applications to receive pasted data from the DOS shield via an INT 16 fast path that bypassed much of the processing a normal INT 9 paste enabled. OS/2 Warp Connect (PowerPC Edition) has a built-in mechanism that allows pasted characters to feed to the application key buffer as fast as the application can retrieve them without overlaying the previous unretrieved characters. Consequently, OS/2 Warp Connect (PowerPC Edition) provides the same function, in the form of a continuous built-in feature while supplying additional function that prevents pasted keystrokes from being overwritten. Hence, this setting is not required.		√	
VIDE0_8514A_XGA_IOTRAP	In OS/2 Warp (Intel), this setting was set to ON to allow controlled access to the video device. It was set to OFF to provide faster unrestricted access for games and graphical applications. IO trapping in OS/2 Warp Connect (PowerPC Edition) has to be done at all times, so there is no need for this foreground performance item.		√	

Table 8 (Page 6 of 6). Changes to the MVM DOS Settings.

DOS Setting	Description	Added	Deleted	Changed
VIDEO_ONDEMAND_MEMORY	This setting was meant to be a way of increasing the speed of the session switch by allocating the shadow VRAM as the access to the physical VRAM was taking place. This setting is no longer supported as it is not clear whether the user can measure the benefits of such a feature or decide if an application would be more usable if the property is set.		√	
VIDEO_RETRACE_EMULATION	This property was meant to improve the performance of the text VRAM updates by emulating the video retrace. This had an adverse effect on the graphical applications.		√	
VIDEO_VRAM_USAGE	This setting controls how the off-screen usage is handled for the DOS sessions.	√		

Glossary

A

Access. An interaction to obtain information from any source or to communicate.

Access control list. A list associated with an object that identifies who can access the object and how they can access the object for example, read-only write-only access.

Access Method. The technique or program used for moving information or communicating.

Access Time. The time from issuing a command to read or write to a file on disk until the physical read or write is actually carried out.

Access Object. Object of the rights administration on which subjects access. Access objects are generally files and interfaces.

Administration Object. Rights administration object which contains rules for several users. The administration object can be assigned to domain, user groups, users, workstations and trusted programs.

Adapter. A part that electrically or physically connects a device to a computer or to another device.

Address. In data communication , the unique code assigned to each device or workstation connected to a network.

AIX. Advanced Interactive eXecutive: IBM variant of Unix operating system.

Albert. 3 Dimensional graphics model developed by Taligent.

Algorithm. A set of rules to solve a problem in a number of steps.

Alias. An alternate label, or name; for example, a name and one or more aliases may be used to refer to the same file.

Alternate personality. An operating mode, or personality, in an environment that supports multiple operating modes, other than the dominant personality. An alternate personality does not control the desktop on display screens. The MVM personality is an example of an alternate personality.

Analog data. Data in the form of physical quantity that is considered to be continuously variable and whose magnitude is made directly proportional to the data or to a suitable function of the data.

Application binary interface. Linkage conventions for a particular microprocessor. They define how registers are used, how they are saved and destroyed across calls, however parameters are passed, stack versus register, and other conventions.

Application program. A program written for or by a user that applies to the user′s work, such as a program that does inventory control or payroll.

API. Application Program Interface. Interface through which programs can communicate.

ASCII. American National Standard Code for information interchange: a coded character set used on personal computers.

Asymmetric multiprocessing. Unequal distribution of tasks across multiple processors.

Asynchronous I/O. I/O operations that are performed separately from the job that requested them.

Authenticate. A process to verify the integrity of data or a message, or to verify the user of an information system or protected source.

Authentication. Confirmation of a given identity, using password, smart card or ID token.

Authentication server. Part of a trusted security base. Responsible for authenticating identities of clients. Maintains passwords and group membership information for users.

Authorize. Granting someone the right to use a computer, application or database; is also used in connection with programs to grant complete or restricted access to an object, resource or function.

Audit. Logging of user actions for audit purposes.

Auditor. Role with regard to rights administration or log evaluation. DP auditor, responsible for auditing DP systems.

B

Base Video Handler. Part of OS/2 Kernel that handles all OS/2 full-screen operations, including both text and graphics in different resolutions.

Base Video Services. Part of OS/2 kernel that provides services to the Presentation Device Driver, Base Video Handler and the Virtual Video Device Driver.

Batch File. On a personal computer, a file having the extension .BAT, which contains a list of commands that are executed when the file is called.

Big endian. See endian.

Bitmap. (1) An area of memory or storage that contains the pixels representing an image, arranged in the sequence in which they are normally scanned, to display the image. (2) A representation of an image by an array of bits.

BIOS. The area of the computer that controls incoming and outgoing signals.

Boot. The process of starting up a personal computer.

Boot Drive. Logical drive from which the operating system is loaded. Generally it is the disk drive (C). It can, however, also be a floppy drive (A).

Boot Protection. Prevention of a system start from a medium other than the hard disk (or boot ROM). A system start with an operating system diskette is prevented.

Bus. In a processor, a physical facility to transfer data; for example, ISA, MCA. Adapter cards are connected to a bus.

Byte. A string that consist of a number of bits, treated as a unit, and representing a character.

C

Card Services. A key element of PCMCIA software architecture. A software management interface that allows you to allocate the system resources (such as memory interrupts) automatically, once the Socket Services detects that a PC Card has been added. Card Services also releases these sources when the PC Card has been removed.

CD-ROM. Compact Disc Read Only Memory. A Compact disc specifically for storing data.

CD-ROM XA. Compact disk read-only-memory extended architecture. A partial implementation of CDI and DVI standards.

Central Service Task. A part of VWIN that contains a list of all Presentation Task and the VDM. It routes the broadcast messages from the VDM or Presentation Task.

Channel. A connection between a personal computer and one or more input/output devices.

Checksum. The sum of a group of data, used for checking purposes.

CID. The IBM architected way to automate installation and customization for Workplace and other products. CID enables LAN connected machines to be installed and maintain remotely.

Client. In the Workplace architecture client-server environment, the consumer of a service. An example is an application or shared service using a client library to communicate with the Workplace Naming Services to retrieve information from the system′s name space.

Client credentials. The set of data associated with a client: user identifier, group identifier(s), roles (i.e. administrator), special permissions etc.

Client Library. A collection of executable personality neutral code and data that is bound to an application and provides the API of a Workplace shared service to clients. The functions of the service API may be implemented in the library or the library may map them to requests to a server or microkernel service.

Client-server model. In Workplace, the Mach defined environment in which a small group of services (servers) at the system layer support a large group of clients (users) at the application layer via interprocess communication.

Client/Server System. A client/server system is a LAN local network.

Clip board. A temporary storage area used to pass information within a program or from a program to another.

Clustered multiprocessing. Distribution of tasks across sets of multiple processors.

CMOS. A chip technology that requires little power, used to store vital configuration data of a PC.

Codepage. An assignment of graphic characters and control function meaning to all code points.

COM. Serial interface for data communication. Is used to connect a modem, for example. Can only be achieved by encryption.

Configuration. The manner in which the hardware and software of an information processing system are organized and interconnected.

Controller. A device that controls the operation of input/output devices.

Conventional Memory. Random Access memory in a PC that DOS or OS uses as the first 640K byte.

Core shared services. A shared service that is includes with OS/2 for PowerPC or Workplace products and can always be counted on to be present. These include name services, file services, pipe services, print/spooler services, loader services, internationalization services, event and windows services, LAN transport services, installation services and software serviceability services.

CRC. Cyclic Redundancy Check. Checksum which is not cryptographic.

D

Device. A physical unit of a computer system, often used for input/output operations, which can be used in a logical order or have a logical address.

DDK. Device Driver Kit; a set of programming tools provided for external device drivers developers.

Device Context. A data structure that is responsible for translating graphics commands made to its associated presentation space into commands that the physical device can convert to displayed information.

Device Driver Interface (DDI). Part of Graphics Engine that is responsible for serializing all the calls from PMGPI.

Directory. A hierarchically structured logical area for storing files on a hard disk or diskette, which may include one or more sub-directories.

Dispatcher. Part of OS/2 Graphics Engine which is responsible for mapping all the functions coming from Device Driver Interafce and dispatch the appropriate functions to the specified driver hardware.

Distributed File System. A file system composed of files or directories that physically reside on more than one computer in a communication network.

Distributed SOM (DSOM). Provides remote access to SOM objects in a transparent way that insulates client programmers from having to have knowledge of the location or platform type where a target object will be instantiated. DSOM allows programmers to use the same object model independently of whether the objects they access are in the same process, in another process on the same machine, or across distributed networks.

Dominant personality. A personality that is started first, provides the desktop (most likely, the desktop is an application of the dominant personality), and exports a set of support functions to alternate personalities and Workplace Shared Services.

DOS. Disk Operating System,: an operating system for personal computers.

Domain. Organizational unit which is commonly managed. Also known as system.

Dynamic Data Exchange (DDE). A messaging protocol that allows PM applications to exchange data via shared memory. DDE transactions always take place between a client application and a server application. The client initiates a dialog with the server by requesting data from the server. The server responds by providing the request data to the client.

Dynamic Link Library (DLL). A file containing executable code and that bound to a program at load time or run tome, rather during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

E

EBCDIC. Extended Binary Coded Decimal Interchange Code: a coded character set used on main-frames.

Emulator. Imitator.

Endian. The addressing model that determines the byte ordering of both data and instructions that are stored in computer memory. The big endian model assigns the lowest address to highest-order or most significant byte of a multibyte scalar data item. The little endian model assigns the lowest address to the lowest order or least significant byte of a multibyte scalar data item.

Event and Window Services. A Workplace core shared service that provides the mechanism for sharing the console device among personalities and applications. It also provides services for the management of window and events.

Event Server. A Part of Event and Window Services (EWS) which is responsible for processing the event input port.

Executable and linking format (ELF). In the Workplace architecture, the object module format.

Extended attribute. The OS/2 method of attaching additional information to a file object. Extended attributes can be used to store notes on file objects (e.g, version, history), categorize file objects (e.g, file type, associations), describe the format of data contained in the file object, or append additional data to the file object. They are stored separately from the file object they are associated with and are managed by the file system attached to the file object.

Extended data. User-defined information, including multimedia information, about Light Table folder objects. Such information goes beyond what is available in OS/2 standard data. Extended data includes user-defined columns, and may come either from a supported database or from extended attributes.

Extension. In the name of a file, the three letters following the dot, which often indicates the type of file, for example, BAT in AUTOEXEC.BAT indicates batch file.

F

Filter GRADD. Extensions for the GRADD that provides a way to modify the GRADDs behavior without rewriting and compiling the GRADD.

First Failure Data Capture (FFDC). A methodology for decreasing the need for the reproduction of user failures by capturing the data associated with a failure as soon as it is detected. The FFDC methodology includes defensive programming, code instrumentation, event tracing, and logging facilities supported by FFST tools.

First Failure Support Technology (FFST). A set of functions that applications can use for problem determination. FFST functions include logging and displaying errors and messages, formatting and routing generic alerts, and generating data dumps.

File Services. A Workplace core shared service that provides a file system framework for multiple logical and physical file systems.

Fileset. The largest unit of a storage device that is managed by Workplace file services. A storage device can be organized into one or more file sets. A fileset is subdivided into directories, which may contain files and other sub directories. Each fileset has a root directory.

File system. In Workplace architecture, software that supports storing data on a storage device. File systems manage the physical locations of data on the storage devices for applications. File systems also manage file I/O operations and control the format of the stored data.

Folder. A directory as represented on the OS/2 desktop.

Font. The characters available for text with a given set of attributes.

Framework. Software package used to provide application programmers with a consistent, easy to use set of services. Framework exports an API for a set of functions that can be provided by multiple software vendors.

Framework service provider. Code that performs the functions associated with a set of service provider interfaces (SPIs). A framework exports an API to client applications. It translates the API to a service provider interface (SPI) that can be supported by multiple service providers. This enables applications to work with service providers from multiple vendors and frees programmers from the need to use a different set of interfaces each time a new product of a given class is introduced.

G

Generation. The number of copies away from the original.

Glyph. A code page has several entries which contain different symbols. These symbols usually include letters, numbers and special characters. Each of these symbols in a code page is called a glyph.

Glyphlist name. The glyphlist name is the name that identifies a set of character glyph names and font index sequence of the character glyph.

Graphics Adapter Device Driver (GRADD). A hardware specific device driver with specific code to exploid the accelerated features of the hardware adapter.

Graphic. Any pictorial representation of information.

Graphical Hardware Interface (GHI). An interface that provides information to the GRADD.

Graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real world scene, often of a desktop. Within that scene are icons representing objects, that the user can access and manipulate with a pointing device.

Graphics (video). Text or pictorial artwork created by a variety of means, such as electronic generated graphics software and the pressed onto the video-discs.

Graphics engine. The drawing engine for a system. It manages display resources such as colormaps and bitmaps. The interface between the graphics transportation protocols and the presentation (display) and printer device drivers.

Graphics Programming Interface (GPI). Part of Presentation Manager which provides the means used by application to do graphics request.

Graphics transportation protocols. The method by which graphics data is communicated between a client (application side) and a server (graphics engine).

H

Hardware Resource Manager. A component of the IBM Microkernel that provides services for configuration , access, and management of hardware to user level device drivers in the Microkernel Services run time environment.

Hertz. A measure of frequency equivalent to cycles per second.

Host. A host is a ″large″ computer which acts as a ″host″ for terminals or workstation PCs with terminal function.

HPFS. High Performance File System. HPFS provides long file name support and fast access to very large disk volumes.

I

Icon. A pictorial representation of a function that you can select to carry out this function.

Identification. Identification of a DP user to the system with a user ID (Name or Personal-Nr).

IFS. Installable file system, the mechanism in OS/2 that permits users to have multiple file systems active at the same time. Installable file systems are loaded during system startup and are attached to storage devices that they have formatted. File system requests for a storage device is directed to the file system that formatted that device.

Interface. Hardware and/or software that links systems, programs, or devices.

Installation Services. A core shared service that provides a common installation facility for Workplace.

Internationalization Services. A Workplace core shared service that provides functions to enable national language support.

Interprocess communication (IPC). In the Workplace architecture, the asynchronous messaging facility for cooperating subsystems to communicate through ports that provides the basis for the Workplace client-server system model.

I/O. Input/Output: pertaining to a device that performs input and/or output operations.

IPL. Initial Program Load; the initialization of a computer.

Image. A still picture or one frame.

Interlace. The technique of using more than one vertical scan to reproduce a complete image. In television, a 2:1 interlace is used, giving two vertical scans per frame. One scan will be odd lines, the other will be even lines.

K

KB. Kilobyte: 1024 bytes.

Kernel. See Microkernel.

Kernel security token. Each client is tagged with a security token in kernel address space. The token is set only by trusted code and represents the user's credentials.

Kilohertz (kHz). Thousands of cycles per second.

L

Local Area Network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication between workstations.

LAN. Local Area Network - local network consisting of server(s), the actual network and PCs as workstations. Safe Guard Professional OS/2 secures LANs.

LAN Transport Services. A Workplace core shared service that provides standard Local Area Network transport layer protocol stacks.

Legacy code. For operating systems, applications developed for earlier environments that must be supported by any new environment.

Link. (1) A logical connection, (2) A physical connection, (3) An interconnection between data or programs.

Linear Executable Format. The object module format used by OS/2 on Intel platforms.

Little endian. See endian.

Loader Services. A Workplace core shared service that provides program loading functions for both personality neutral and personality dependent tasks.

Logical File System. A component of Workplace File Services that provides a consistent view of multiple Physical File Systems. The Logical File System provides path resolution and other services that are independent of the on-disk format of data.

LPT. Parallel interface to attach a printer, streamer, etc.

M

Machine readable information. Text information subject to international translation considerations.

Message Interface Generator. A Workplace tool that generates low level remote procedure calls between a client and server process using the microkernel IPC mechanism. The code that is generated includes routines to pack and unpack the messages used to communicate between processes.

Message Queue. Part of PM application that contains all the messages coming from System Input Queue.

Microkernel. In the Workplace architecture, the component of the IBM Microkernel Product that runs in the most privileged state of the computer and controls the basic operation of the machine. It includes only those functions required to provide a set of abstract processing environments and to permit applications to work together as clients and servers. These environments are IPC, ports, tasks, threads, and virtual memory management.

Microkernel Services. The system services provided with the IBM microkernel product. There are three classes of services: Kernel services, Shared services Device services.

Migrate. (1) To move data from one storage media to another, (2) To change to a new operating environment.

Module. Program module which takes over a specific function. Example: logging in a linear file on the server, or logging in a local ring buffer file. Modules can be swapped by the system administrator.

Multi-user. Pertaining to serial or concurrent use by more than one user. Capable of distinguishing between different users and assigning individual user ownership of information.

Multi-tasking. A technique that allows several processes to appear to run simultaneously, even though the computer only has one CPU. This is achieved by sequentially switching the CPU between tasks.

Multiplexer. A device that interleaves the transmission of several input signals over a connection such that the input signals can be recovered.

Multiprocessor. A processing unit consisting of two or more independent processors acting as parallel.

MVM Personality. The Workplace personality that manages the collection of Virtual x86 machines. The MVM personality provides DOS/Windows/DPMI program compatibility on a PowerPC. It manages and supports execution of multiple x86 DOS and Windows programs (in real mode and protected mode) each in its own Virtual x86 machine.

N

Name Services. A core shared service that provides a programming interface for its client to store and access transient or persistent system and state information in the global name space. It exports APIs to its clients and defines SPIs for name service providers.

Name Services entry attribute. Entry information required by Workplace Name Services for each object it manages. It consists of three descriptors: class identifier, attribute name, and attribute value.

Name Services mount. A point or node in a client′s name space tree established by a name service provider and implemented as a sub tree in the name space.

Name space. A hierarchical (tree) structure of nodes, with one distinguished node called the root. Each node is labeled by a distinct, non-empty string of characters, called a

component name. Nodes are directories or leaves. Leaves are terminal nodes that represent an entity identified by a mach port: file, device, root of a fileset, user name. Each node may have a list of attributes and access control. A name space is assigned naming rules during creation.

Network. (1) A network of devices and software connected for information interchange, (2) An arrangement of modes and connecting branches to interconnect computers, terminals and workstations.

National Language Support (NLS). The modification or conversion of a United States English product to conform to the requirements of another language or country. This cam include the enabling or retrofitting of a product and the translation of nomenclature, MRI or documentation of a product.

Node. In a network , a point at which one or more units are connected. Each node has a network address.

O

Object. (1) Resource of the DP system, such as files, interfaces, networks, etc. (2).A visual component of a user interface that a user can work with to perform a task.

Object linking and embedding (OLE). An application protocol established by Microsoft Corp. that allows objects created by one application to be linked to or embedded in objects created by another application.

Object module. A binary executable or data component or both resulting from source assembly or compilation, or from secondary linkage of such object modules.

OEM. Equipment sold by another manufacturer.

OS2CHAR. A component of the OS/2 Presentation Manager which processes input for OS/2 sessions. It provides a common set of video code to implement both full screen and windowed VIO.

P

Panel. The set of information displayed on the screen of a display station.

Password. In computer security, a string of characters used to gain access to a computer file or system, during sign-on or at a later time. A PIN can be considered as a password.

Path. (1) In a network, any route between two nodes, (2) The route traversed by information exchanged between two network devices, (3) A command in DOS related to the path through its (sub)directory structure to reach a file.

Path resolution. The resolution of a path to the correct file services entity (directory or file).

Pause Function. ″Logoff″ for a short work interruption. The screen is blanked, the keyboard can only be used for special entries and the work station is locked. To continue work, the user who triggered the pause, must log on again.

Personal Computer Memory Card International Association (PCMCIA). A non profit technical standards and trade association established to develop a common format for integrated circuit PC Cards. PCMCIA standards describe the physical requirements, electrical specifications, and software architecture for these cards. The key elements of the PCMCIA software architecture are Socket Services and Card Services.

Pel. Picture element. The smallest building block that a screen or bit-mapped image can display. Pel and pixel can be used interchangeably.

Personality. An operating system dependent application execution environment that corresponds to a traditional operating system such as OS/2, DOS, or AIX.

Personality dependent (PD). Code that depends on APIs or services provided by a Workplace personality service.

Personality neutral (PN). Code that is operating system independent because it depends only on APIs provided by the IBM microkernel product or other microkernel services and does not use any APIs or services provided by a personality service.

Pipe services. A Workplace core shared service that implements named and unnamed pipes with OS/2 semantics.

Pixel. A single point of an image, having a single pixel value.

Physical File System (PFS). A Workplace component that manages the on disk storage, indexing, mounting, and recovery of data. The File Services framework supports the installation of multiple Physical File Systems. Examples of Physical File Systems iclude FAT, HPFS, and CDROM.

PM Shield. Part of PM that is resnposible of impelementing a Presentation Manager window for the session. It manges the user interface of each of the windows it supports. It contains the window procedure for this window.

PMI File. A file that contains data and commands necessary to provide support for modes beyond VGA in a non-BIOS environment. The information is used by VIDEOPMI to support display adapter.

PMWIN. Part of Presentation Manager which is responsible for creating, maintaining and destroying windows on the PM desktop.

Pop-up. A window which appears on the screen to display text, graphics, messages, or documents.

Port. In the Workplace architecture, a unidirectional asynchronous communication channel between a client and a server. A port has a single receiver and may have multiple senders. If a reply is provided to a service request a second port must be used.

Power Management. A software subsystem that extends battery life in portable computer systems and conserves electrical energy for all non-battery powered systems.

Presentation device drivers. Device drivers that process the high level function calls to the Presentation Manager interface and communicate with physical device drivers or the display hardware.

Presentation Manager (PM). The Workplace services that presents a graphics based interface to applications.

Presentation device drivers. Device drivers that process the high level function calls to the Presentation Manager interface and communicate with physical device drivers or the display hardware.

Presentation Driver (PD). Device-dependent tools used by Graphics Engine to map its graphics layout. Presentation Driver will be different for every hardware supported.

Presentation Manager (PM). The Workplace services that presents a graphics based interface to applications.

Presentation Space (PS). A data structure which contains the device-independent output. This is the place which graphic images are created before being sent to an output device.

Protocol. Rules and agreements for communication between devices.

R

RAM. Random Access Memory: Memory where data can be written and read directly.

Reuse. This is the recreation of the original status of a file, the main memory or the swap file after it has been deleted or after the user has logged off.

Resolution. The ability of an image reproducing system to reproduce fine detail.

Role. Role which the user plays, particularly with regard to rights administration. A role is assigned specific administrative rights, e.g. system administrator, auditor, accessory or simple user.

ROM. Read Only Memory; Memory to store programs or data permanently.

S

Scanner. A device which performs scanning.

Schema. The data-definition part of a database table.

Seamless Windows. A Windows application that runs in OS/2 desktop, side by side with OS/2 applications and other Windows applications.

Server. On a LAN, a station that provides services to other stations; for example, file server, print server, and security server.

Service manager. Installation programs, remote administrators, or other types of clients that can interact with the state of frameworks and service providers through the Name Services.

Session. The period of time that a network connection lasts, including the establishment and release of the connection.

Shared service. A client library that exports an API to personality neutral and personality dependent client applications and an optional server component. The server may be personality dependent.

Software Configuration Utility. One of the utilities distributed with the workstation security services program for configuring security servers and device drivers.

Software Drawing for Non-Accelerated Graphic Operations (SOFTDRAW). A generic graphics library used for software simulation.

Sub-directory. A directory contained within another directory in the file system hierarchy.

Subject. A subject is a user or process which accesses objects (files etc.).

Subsystem. A secondary or subordinate system, usually capable of operating independently.

T

Token. Bit string (combination of bits) to enable the execution of a specific operation.

Token-ring. An IBM network with a ring topology that passes tokens from one attaching device to another.

Translation layers (GRE2VMAN,GDI2VMAN). A layer that exists between the Graphics Engine and the Video Manager used for translation of engine commands to VMI commands.

U

Unicode. Code that is independent of language and culture, supports multiple simultaneous character sets.

User. User in a system.

Userid. User identification, name for a user in the system.

V

Video Manager (VMAN). A component used as a central point of distribution, for coordinationg, serializing and dispatching certain commands to different components of the GRADD model.

Video Manager Interface (VMI). An Interface that provides information to the Video Manager.

VIDEOPMI. A shared madule which communicates to/from the protected mode video device driver (BVHSVGA) as well as the virtual video device driver (VSVGA).

VIO API. Set of functions that is used to create application in OS/2 full-screen.

Virtual Device Driver. A module that virtualizes hardware and ROM BOS services on a per-VDM basis. They provide support for direct manipulation of memory-mapped I/O devices, and the direct programming of I/O ports.

Virtual DOS Machine (VDM). The place that each DOS application runs in v86 mode. It is a v86-mode variant of an OS/2 single-thread process. Each VDM executes a DOS application and emulates the functions of DOS in a virtual PC environment.

Vital product data. Data that describes the hardware and software parts of the system.

Virtual Video (VVIDEO). A component of Multiple Virtual Machine (MVM) that is responsible for routing all the video requests to hardware.

Virtual Video Device Driver. It is used by DOS applications which are running in a DOS session. It will provide direct manipulation of memory-mapped video I/O devices. It manages all access to the video memory, registers and video ROM BIOS.

Virtual Windows (VWIN). A virtual device driver that allows a Windows program to run on the OS/2 desktop. It is the link that passes messages from on GUI to another so that both environments can know about and adjust for each other.

W

Wildcards. Placeholders for any number of other characters. An asterix (*) stands for a permitted set of any other characters. A question mark (?) stands for any other single character.

Window Procedure. Part of PM applications which is responsible for managing messages coming to the window object.

Windows Seamless Device Driver. It is a standard Windows display device driver. It is derived from the standard Windows driver by conditional compile

WinShield. It is the Windows counterpart of PMShield. WinShield serves a complementary purpose, maintaining Workplace Shell windowing state information for its VDM.

Workstation. A terminal or microcomputer that often is connected to a main frame or a network, at which the user can perform applications.

X

XGA. Extended graphics array. A high resolution display with a display matrix (pels) of 1,024 x 768 at 256 colors. XGA can also provide more colors with reduced resolution (640 x 480 at 65,536 colors).

List of Abbreviations

ABI	Application Binary Interface
ACL	Access Control List
ADC	Analog-to-Digital Converter
AIX	Advanced Interactive eXecutive
ANSI	American National Standards Institute
APAR	Authorized Program Analysis Report
API	Application Programming Interface
AT	Advanced Technology
AVSS	Audio-Visual Sub System
BGA	Business Graphics Adapter
BIDI	Bi-directional
BIOS	Basic Input Output System
BMP	Bit-Mapped Graphics
BVH	Base Video Services
CAE	Common Application Environment
CD	Compact Disc
CD-ROM	Compact Disk - Read-Only Memory
CGA	Color Graphics Adapter
CID	Configuration, Installation, and Distribution
CORBA	Common Object Request Broker Architecture
COSE	Common Operating System Environment
CPU	Central Processor Unit
DDE	Dynamic Data Exchange
DDK	Developer Driver Kit
DFS	Distributed File System
DIB	Device Independent Bitmap
DLL	Dynamic Link Library
DMA	Direct Memory Access
DOS	Disk Operating System
DPI	Dots Per Inch
DSOM	Distributed SOM
DSP	Digital Signal Processor
DVI	Digital Video Interactive
EA	Extended Attribute
EGA	Enhanced Graphics Adapter
EISA	Extended Industry Standard Architecture
ELF	Executable and Linking Format
EnDive	Enhanced Direct Interface to Video Extension
FAT	File Allocation Table
FFDC	First Failure Data Capture
FFST	First Failure Support Technology
GDI2VMAN	Graphics Design Interface to Video Manager
GHI	Graphics Hardware Interface
GIF	Graphics Interchange Format
GPIB	General Purpose Interface Bus (IEEE 488)
GRADD	Graphics Adapter Device Driver
GRE2VMAN	Graphics Engine to Video Manager
GUI	Graphical User Interface
HPFS	High Performance File System
IBM	International Business Machines

IEEE Institute of Electrical and Electronical Engineers

IFF Interchange File Format

IFI Intelligent Font Interface

IFS Installable File Systems

IOCA Image Object Content Architecture

IPC Interprocess Communication

IRQ Interrupt Request

ISA Industry Standard Architecture

ISDN Integrated Services Digital Network

ISO International Organization for Standardization

ISV Independent Software Vendors

Kbps Kilobytes per second

kHz Kilohertz

LAN Local Area Network

LSB Least Significant Bit

LX Linear Executable Format

M-O Magneto-Optical

MB Megabyte (1,048,576 bytes)

Mbps Megabits per second

MBps Megabytes per second

MC Micro Channel

MCA Micro Channel Architecture

MCD Media Control Driver

MCGA Modified Color Graphics Adapter

MIDI Musical Instrument Digital Interface

MIG Message Interface Generator

MME Multimedia Extension

MMIO Multimedia I/O Services

MMPM/2 Multimedia Presentation Manager/2

MRI Machine Readable Information

MSB Most Significant Bit

MTC Midi Time Code

NLS National Language Support

OEM Other Equipment Manufacturer

OLE Object Linking and Embedding

OS/2 Operating System/2

PC Personal Computer

PC AT Personal Computer Advanced Technology

PC XT Personal Computer Extended Technology

PCMCIA Personal Computer Memory Card International Association

PD Personality Dependent

PD Presentation Driver

PEL Picture Element

PFS Physical File System

PM Presentation Manager

PM Presentation Manager

PMDD Presentation Manager Display Driver

PMGPI Presentation Manager Graphics Programming Interface

PMGRE Presentation Manager Graphics Engine

PMI Protect Mode Interface

PMVDD Presentation Manager Video Device Driver

PMWIN Presentation Manager Window

PN Personality Neutral

POSIX Portable Operating System Interfaces for Computer Environments

PS/1 Personal System/1

PS/2 Personal System/2

R/W Read/Write

RAM	Random Access Memory
RAS	Reliability, Availability, Serviceability
REXX	Restructured Extended Executor language
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RPQ	Request for Price Quotation
SCSI	Small Computer Systems Interface
SDK	Software Developers Kit
SMP	Symmetric Multi Processing
SOM	System Object Module
SPI	Service Provider Interface
SQL	Structured Query Language
SVGA	Super Video Graphics Adapter
TCB	Trusted Computing Base
VDH	Virtual Device Helper
VDM	Virtual DOS Machine
VGA	Video Graphics Adapter
VMAN	Video Manager
VMI	Video Manager Interface
VPD	Vital Product Data
VRAM	Video Random Access Memory
VVMI	Virtual Video Manager Interface
VWIN	Virtual Windows
WORM	Write Once Read Many
WPS	Work Place Shell
XGA	Extended Graphics Array
XMA	Expanded Memory Array
XMS	Extended Memory Specification
XOM	Extended Open Management

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

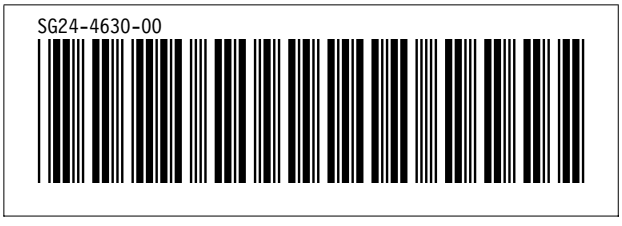
V

W

X

Z

Printed in U.S.A.



Artwork Definitions

Table Definitions

Figures

Headings

Index Entries

Tables

Processing Options

Runtime values:

Document fileid	SG244630 SCRIPT
Document type	USERDOC
Document style	SDELIB
Profile	EDFPRF40
Service Level	0022
SCRIPT/VS Release	4.0.0
Date	95.12.28
Time	09:59:56
Device	3820A
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR G	INLINE
SYSVAR V	ITSCEVAL
SYSVAR X	YES

Formatting values used:

Annotation	NO
Cross reference listing	YES
Cross reference head prefix only	NO
Dialog	LABEL
Duplex	YES
DVCF conditions file	(none)
DVCF value 1	(none)
DVCF value 2	(none)
DVCF value 3	(none)
DVCF value 4	(none)
DVCF value 5	(none)
DVCF value 6	(none)
DVCF value 7	(none)
DVCF value 8	(none)

DVCF value 9	(none)
Explode	NO
Figure list on new page	YES
Figure/table number separation	YES
Folio-by-chapter	NO
Head 0 body text	Part
Head 1 body text	Chapter
Head 1 appendix text	Appendix
Hyphenation	NO
Justification	NO
Language	ENGL
Keyboard	395
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	4
Partial TOC (new page after)	INLINE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	.,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	YES
Title page (draft) alignment	RIGHT
Write cross-reference file	(none)

Imbed Trace

Page 0	4630SU
Page 0	4630VARS
Page 0	4630FM
Page i	4630EDNO
Page ii	4630ABST
Page xiii	4630SPEC
Page xiii	4630TMKS
Page xiv	4630PREF
Page xviii	4630ACKS
Page xx	30INTRO
Page 3	30MK
Page 30	30SERV
Page 31	30DS
Page 32	30EWS
Page 40	30FS
Page 47	30PIPE
Page 48	30OS2
Page 51	30CP
Page 56	30CP1
Page 65	30CP2
Page 72	30MVM
Page 86	30GUI
Page 119	30PRT
Page 122	30SYS
Page 128	30INSTA
Page 139	30APS
Page 142	30APP1
Page 148	4630GLOS
Page 160	4630ABRV
Page 178	4630EVAL