

IBM Copyright Permission #22527

**Reprint Courtesy of International Business Machines Corporation, © 1994
International Business Machines Corporation'**

INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM) ARMONK, NEW YORK 10504

PERMISSION TO REPRINT/POST IBM COPYRIGHTED PUBLICATIONS

The material owned by IBM must be accompanied by the following credit line: **"Reprint Courtesy of International Business Machines Corporation, © [Year] International Business Machines Corporation"**. The credit line normally should appear on the page where the posting appears, either under the title or as a footnote. If the foregoing is inconvenient, the credit line may be placed in a conveniently viewable manner with suitable reference to the places where the material appears.

It is the understanding of **International Business Machines Corporation** that the purpose for which its material is being reproduced is accurate and true as stated in the original request.

Permission to quote from, transmit electronically or reprint/post IBM material is limited to the purpose and quantities originally requested and must not be construed as a blanket license to use the material for other purposes or to reproduce other IBM copyrighted material.

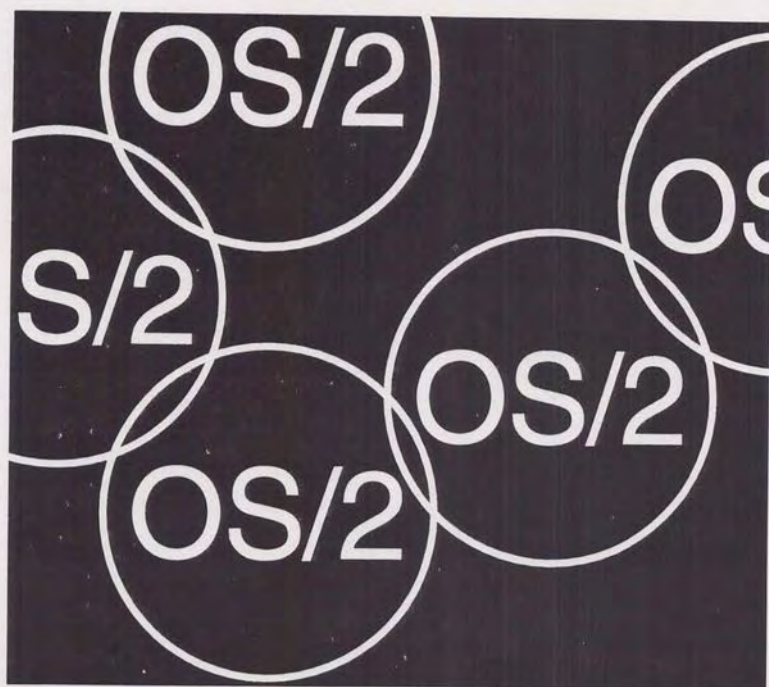
IBM reserves the right to withdraw permission to reproduce copyrighted material whenever, in its discretion, it feels that the privilege of reproducing its material is being used in a way detrimental to its interest or the above instructions are not being followed properly to protect its copyright.

No permission is granted to use trademarks of **International Business Machines Corporation** and its affiliates apart from the incidental appearance of such trademarks in the titles, text, and illustrations of the named publications. Any proposed use of trademarks apart from such incidental appearance requires separate approval in writing and ordinarily cannot be given. The use of any IBM trademark should not be of a manner which might cause confusion of origin or appear to endorse non-IBM products.

THIS PERMISSION IS PROVIDED WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

INTERNATIONAL BUSINESS MACHINES CORPORATION

Dated: April 22, 2014



OS/2 for Software Developers

Version 2.x

(CN17400C/N1740)

Volume 1

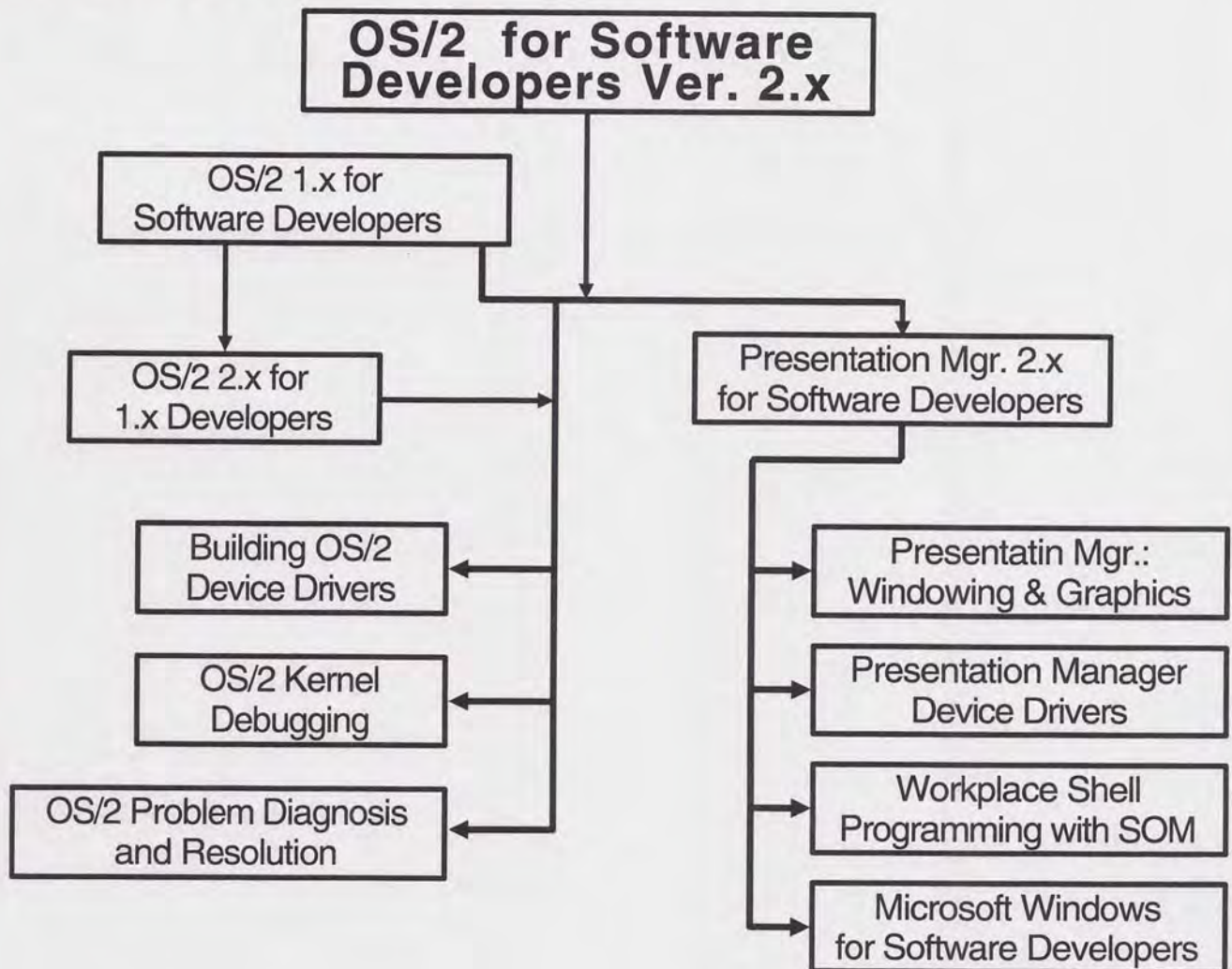
Class Workbook

~~HeapMem~~ - cleans up unused/lost storage
use: view ddec4lib -heapmin

Course materials may not be reproduced in whole or in part without the prior written permission of SKILL DYNAMICS - An IBM Company

© IBM Corporation (1991 - 1993)
All Rights Reserved

Course Map



CN17410C/N1741	Microsoft Windows for Software Developers
CN17400C/N1740	OS/2 for Software Developers Ver. 2.x
CN17500C/N1750	Building OS/2 Device Drivers
CN16880C/N1688	Presentation Manager 2.x for Software Developers
CN17450C/N1745	Workplace Shell Programming with SOM
CN17440C/N1744	Presentation Manager: Windowing & Graphics
CN17420C/N1742	Presentation Manager Device Drivers
CN17430C/N1743	OS2 Problem Diagnosis and Resolution
CN17460C/N1746	OS/2 2.x for 1.x Developers
WTDMP10S	Introduction Video (SRA)
WTDMP20C	80x86 Assembler and Base Architecture
WTDMP30C	Interfacing the Personal Computer
WTDMP50C	80386 Hardware Architecture
WTDMP60C	80386/486 Assembler and Architecture
WTDMP70C	80486 Hardware Architecture

Contact Bob Rohr, Skill Dynamics (ROHR @ DALVM1)
914/742-5653 or Tie-line 770-5653

This page intentionally blank

OS/2 for Software Developers Ver. 2.x

[CN17400C/N1740]

5 days Hands-on Labs

Public, Private

This course combines lectures, demonstrations and laboratory exercises to present the primary functions of the OS/2 2.0 control program. Daily exercises provide the student with experience in implementing the OS/2 2.0 Application Programming Interface.

Who Should Take This Course: Systems programmers and engineers who will use OS/2 as a base for developing hardware and software products. Persons experienced in programming OS/2 version 1.x should attend the OS/2 2.0 for OS/2 Software Developers course (WTDPS25C) for an update.

What You Are Taught: The student will be able to apply the appropriate programming tools and procedures to build OS/2 applications that implement these key features:

- OS/2 Design Goals
- Application Types
- 80386/486 Protected Mode Operation
- Memory Management
- Thread Management
- Execution Synchronization Using Semaphores
- Process Management
- Interprocess Communication Using Memory/Pipes/Queues
- File Systems
- Dynamic Link Libraries
- Session Management
- Exception Management

Prerequisites: Familiarity with the PS/2 operating and programming environment and a working knowledge of the C programming language.

Course Outline

Day 1 (A.M.)

Introduction to OS/2 [45 minutes]

- Design Goals of OS/2
- Memory Organization
- Privilege Model
- Application Programming Interface
- Multitasking Hierarchy

Pre-course Quiz [10 minutes]

OS/2 Application Types [30 minutes]

- OS/2 Mode Windowed Applications
- OS/2 Mode Full-Screen Applications
- DOS/Windows Mode Applications

Protected Mode [90 minutes]

- General Purpose and Memory Registers
- Operating Modes
- Protection Mechanisms
- Real Mode Addressing
- Protected Mode Addressing
- Memory Exceptions

Day 1 (P.M.)

Introduction to OS/2 Programming [30 minutes]

- Register-Based API
- Call-Based API
- C Program Template
- Function Prototypes
- Return Codes

Environment Strings and Command Line Arguments [45 minutes]

- Obtaining Environment and Command Line Pointers
- Searching the Environment
- Environment Strings
- Command Line
- C Program Entry State

Program Development [30 minutes]

- Programming Tools
- Header and Include Files
- Development Process
- LINK386
- NMAKE

Laboratory Exercise 1 [180 minutes]

- Introduction to the Enhanced Editor
- Writing an OS/2 Program
- IBM Presentation Debugger
- System Exceptions/Faults
- Displaying Command Line Arguments
- Experimenting with DETACH and START
- WorkFrame/2

Day 2 (A.M.)

Memory Management [70 minutes]

- OS/2 2.0 Memory Model
- Virtual Address Space
- Memory Objects
- Memory Allocation API's
- Memory Suballocation API's

Introduction to Multitasking [20 minutes]

- Serial Multitasking
- Parallel Multitasking
- Elements of Multitasking
 - Sessions
 - Processes
 - Threads

Thread Management [90 minutes]

- Thread Hierarchy
- Creating Threads
- Thread States
- Thread Priority
- Configuration Parameters
- C Programming Considerations

Day 2 (P.M.)

Laboratory Exercise 2 [180 minutes]

- Building a Single Thread OS/2 Application
- Multitasking Using Two Threads
- Thread Control
- Thread Management Using C Functions

Day 3 (A.M.)

Synchronizing with Semaphores [90 minutes]

- Uses of Semaphores
- Classes and Types of Semaphores
- Application of OS/2 Semaphores

- Event Semaphore API's
- Mutex Semaphore API's
- MuxWait Semaphore API's

OS/2 Timer Services [20 minutes]

- Synchronous Timer API's
- Asynchronous Timer API's

Process Management [70 minutes]

- Process Resources
- Creating A Process
- Process Isolation
- Testing Process Status
- Controlling Process Termination

Day 3 (P.M.)

Shared Memory Objects [40 minutes]

- Named Shared Objects
- Unnamed Shared Objects
- Shared Memory Allocation API's

Laboratory Exercise 3 [180 minutes]

- Synchronizing Threads With Event Semaphores
- Creating and Managing Processes
- Synchronizing Parent/Child Processes
- Exit Routines

Day 4 (A.M.)

File Input/Output [60 minutes]

- OS/2 File Systems
- Naming OS/2 Files
- Open/Creating Files
- Standard File Attributes
- Extended File Attributes

Interprocess Communication Using Queues [45 minutes]

- Client/Server Interprocess Communication
- Queue Management API's

Dynamic Link Libraries [90 minutes]

- Static Linking
- Dynamic Linking
- Code Sharing
- Building a DLL
- Methods of Dynamic Linking
- DLL Initialization

Day 4 (P.M.)

Laboratory Exercise 4 [180 minutes]

- Load-Time Dynamic Linking
- Updating a DLL
- Run-Time Dynamic Linking
- DLL Initialization
- Pipes and Queues

Day 5 (A.M.)

Exception Management [90 minutes]

- System Exceptions
- Types of Exceptions
- Exception Handlers
- Exception Management Data Structures
- Exception Management API's

Session Management [45 minutes]

- Session Management Components
- Session Hierarchy
- Session Management API's

Interprocess Communication Using Pipes [30 minutes]

- Standard Input/Output
- Anonymous Pipes
- Named Pipes

Post-Course Quiz [30 minutes]

Day 5 (P.M.)

Laboratory Exercise Completion [open]

References:

ADG - Application Design Guide, S10G-6260
C.P.- Control Programming Reference, S10G-6263
G.S. - Getting Started, IBM Developer's Toolkit for OS/2 2.0, 10G6199
P.G.1 - Prog.Guide Vol. 1, OS/2 Tech. Lib, S10G-6264
U.G. - C Set/2 User's Guide, S10G-4444
Kogan - The Design Of OS/2, Kogan & Deitel, Addison-Wesley
Letwin - Inside OS/2, Gordon Letwin, Micorsoft Press

1. Introduction to OS/2

- 1 Introduction
- 2 Design Goals - Multitasking, DevIndependence Letwin 9
- 3 Design Goals - Custom and Stable Environment Letwin 15
- 4 Design Goals - 32 Bit Environment
- 5 Design Goals - DOS/Windows Support
- 6 OS/2 2.0 Memory Organization
- 7 OS/2 Privilege Model Kogan 37, ADG 2-11
- 8 Application Programming Interface ADG 1-29
- 9 Layers of OS/2 Kogan 86
- 10 OS/2 Multitasking Hierarchy Kogan 87

2. OS/2 Applications

- 1 Introduction
- 2 Types of Applications
- 3 OS/2 Mode Window Applications
- 4 OS/2 Mode Full Screen Applications
- 5 DOS /Window Mode Applications
- 6 Concurrent Execution of Application Classes
- 7 System Limits - 2.0 vs 1.3

3. Protected Mode

- 1 Introduction
- 2 General Purpose Registers Kogan 29,43
- 3 Memory Segment Registers Kogan 43
- 4 Operating Modes
- 5 Protection Mechanisms
- 6 Real Mode Memory Addressing
- 7 Protected Mode Memory Addressing
- 8 Selector Format
- 9 Data Segment Descriptors
- 10 16-Bit Protected Mode Memory Addressing
- 11 32-Bit Protected Mode Memory Addressing
- 12 FAULT.LST
- 13 First Violation - Part 1
- 14 First Violation - Part 2
- 15 First Violation - Part 3
- 16 Second Violation
- 17 Third Violation
- 18 Fourth Violation

4. Intro to OS/2 Programming

- 1 Introduction

-2	OS/2 Application Interface	
-3	DOS Register-based API	
-4	OS/2 Call-Based API	
-5	Using the 'C' Language for OS/2	
-6	API Function Prototypes	
-7	_System Calling Convention	
-8	Return Codes	
-9	Class Exercise	
5.	Environment Strings and Command Lines	
-1	Introduction	
-2	Obtaining Environment and Command Line Pointers	
-3	Searching the Environment	
-4	Environment Strings	
-5	Command Line and Arguments	
-6	'C' Program Entry State	
-7	DosScanEnv Program Example	
6.	Software Development Process	
-1	Introduction	
-2	Programming Tools	G.S. Chp. 4
-3	Header and Include Files	
-4	OS/2 Development Process	
-5	OS/2 Linker	
-6	NMAKE Program Maintenance Utility	
7.	Memory Management	P.G.1 Chp. 6
-1	Introduction	
-2	The Flat Memory Model	Kogan 163
-3	Virtual Address Space	Kogan 164
-4	Memory Objects	Kogan 165
-5	Types of Memory Objects	Kogan 165,166
-6	DosAllocMem	C.P.2-6
-7	DosFreeMem	C.P. 2-115
-8	DosQueryMem	C.P. 2-222
-9	DosSetMem	C.P. 2-317
-10	Suballocation of Memory	P.G.1 6-14
-11	DosSubSetMem	C.P. 2-361
-12	DosSubAllocMem, DosSubFreeMem	C.P. 2-357,359
-13	DosSubUnsetMem	C.P. 2-364
8.	Intro to Multitasking	
-1	Introduction	
-2	Multiple Tasks	Letwin 41
-3	Parallel Multitasking	Letwin 13, 42
-4	OS/2 Elements of Multitasking (Sessions)	
-5	OS/2 Elements of Multitasking (Processes)	Letwin 44
-6	OS/2 Elements of Multitasking (Threads)	Letwin 43, 70
9.	Thread Management	P.G.1 Chp. 7
-1	Introduction	
-2	Thread Hierarchy	
-3	An Example of Three Threads	
-4	Another Example of Three Threads	
-5	Thread States	
-6	Priority Classes and Levels	Kogan 116
-7	Dynamic Priority Adjustment	Kogan 117

-8	Configuration Parameters Effecting Multitasking	
-9	Thread Management Functions	
-10	DosGetInfoBlocks	C.P. 2-129
-11	DosCreateThread	C.P. 2-53
-12	DosEnterCritSec, DosExitCritSec	C.P. 2-78, 97
-13	DosSuspendThread, DosResumeThread	C.P. 2-366, 281
-14	DosKillThread	C.P. 2-149
-15	DosWaitThread	C.P. 2-386
-16	DosSetPriority	C.P. 2-327
-17	Mutual Exclusion Problems	
-18	Variables in 'C' Programs	
-19	Reentrancy with Multiple Threads	
-20	Reentrancy with C Library Functions	
-21	_beginthread/_endthread C Library Function	
-22	DosCreateThread vs _beginthread ()	
10.	Synchronization Using Semaphores	P.G.1 Chp. 8
-1	Introduction	
-2	Uses of Semaphores	P.G.1 8-1
-3	Classes and Types of Semaphores	P.G.1 8-1
-4	Application of OS/2 Semaphores	
-5	DosCreateEventSem	C.P. 2-38
-6	DosOpenEventSem	C.P. 2-164
-7	DosCloseEventSem	C.P. 2-24
-8	DosPostEventSem, DosResetEventSem	C.P. 2-184, 279
-9	DosWaitEventSem	C.P. 2-379
-10	DosQueryEventSem	C.P. 2-206
-11	DosCreateMutexSem	C.P. 2-40
-12	DosOpenMutexSem	C.P. 2-166
-13	DosCloseMutexSem	C.P. 2-25
-14	DosRequestMutexSem, DosReleaseMutexSem	C.P. 2-273, 272
-15	DosQueryMutexSem	C.P. 2-233
-16	Critical Section Vs MutexSem	
-17	DosCreateMuxWaitSem	C.P. 2-42
-18	DosOpenMuxWaitSem	C.P. 2-168
-19	DosCloseMuxWaitSem	C.P. 2-26
-20	DosAddMuxWaitSem, DosDeleteMuxWaitSem	C.P. 2-4, 63
-21	DosWaitMuxWaitSem	C.P. 2-381
-22	DosQueryMuxWaitSem	C.P. 2-235
-23	Summary of Semaphore API's	
11.	OS/2 Timer Services	P.G.1 Chp. 1
-1	Introduction	
-2	DosSleep	C.P. 2-341
-3	DosStartTimer	C.P. 2-351
-4	DosAsyncTimer	C.P. 2-12
-5	DosStopTimer	C.P. 2-353
12.	Processes	P.G.1 Chp. 7
-1	Introduction	
-2	Processes	Kogan 87
-3	Process Resources	Letwin 44
-4	Creating a Process	
-5	DosExecPgm	C.P. 2-89
-6	Process Isolation	
-7	Controlling Child Processes	
-8	Process Concurrency	

-9	Command Subtrees	Letwin 57
-10	Testing for Child Process Termination	P.G.1 2-375
-11	DosWaitChild: Process Subtrees	Letwin 64,65
-12	DosWaitChild: Individual Processes	
-13	DosWaitChild: Process Subtrees	
-14	Performing Cleanup When a Process Ends	
-15	Exit List Sequence	
-16	DosExitList	P.G.1 2-98
13. Shared Objects		P.G.1 6-18 thru 20
-1	Introduction	
-2	Two Types of Shared Objects	
-3	Using Named Shared Segments	
-4	Using Unnamed Shared Segments	
-5	DosAllocSharedMem	C.P. 2-9
-6	DosGetNamedSharedMem	C.P. 2-135
-7	DosGiveSharedMem	C.P. 2-141
-8	DosGetSharedMem	C.P. 2-139
14. File Input/Output		P.G.1 Chp. 2 thru 5
-1	Introduction	
-2	OS/2 File Systems	P.G.1 2-1
-3	Multitple OS/2 File Systems	P.G.1 2-2
-4	Recognizing Dos and OS/2 file	P.G.1 2-5
-5	Naming OS/2 Files	P.G.1 3-1
-6	Using Long File Names	P.G.1 3-3 thru 5
-7	Opening/Creating Files	
-8	DosOpen	C.P. 2-158
-9	Synchronous File Input/Output	C.P. 2-265, 388
-10	Asynchronous File Input/Output	
-11	Changing the Size File	C.P. 2-312
-12	Inheriting Files	
-13	Locking Regions of a File	C.P. 2-304
-14	Updating File Data/Directory	P.G.1 4-5, C.P. 2-22, 277
-15	Standard File Attributes	P.G.1 5-1
-16	Extended File Attributes	P.G.1 5-1
-17	Naming Extended Attributes	P.G.1 5-2
-18	Extended Attribute Data Types	P.G.1 5-3
-19	Extended Attribute Data Structures	P.G.1 5-15
-20	Example of a Single-Value EA	
-21	Example of a Multi-Value, Single-Type EA	
-22	Example of a Multi-Value, Multi-Type EA	
-23	Preserving EAs	P.G.1 5-16
-24	Managing EAs	
15. Queues		P.G.1 Chp. 10
-1	Introduction	
-2	Queue Overview	
-3	DosCreateQueue	C.P. 2-51
-4	DosOpenQueue	C.P. 2-170
-5	DosWriteQueue	C.P. 2-391
-6	DosPeekQueue	C.P. 2-177
-7	DosReadQueue	C.P. 2-268
-8	DosQueryQueue,DosPurgeQue,DosCloseQueue	C.P. 2-254, 186, 28
-9	Monitoring Multitple Queues	
16. Dynamic Link Libraries		

-1	Introduction	
-2	Static Linking	
-3	Disadvantages of Static Linking	
-4	Advantages and Disadvantages of Dynamic Link Libraries	
-5	Dynamic Links vs Processes	
-6	Dynamic Linking	
-7	Load Time	
-8	Code Sharing: Two Instances of Same .EXE	
-9	Code Sharing: Different .EXE's with Common Subsys	
-10	Code Sharing: Different .EXE's, DLL with Static Data	
-11	Code Sharing: Different .EXE's, DLL with Instance Static Data	
-12	Variables in Main and in the DLL	
-13	Building a DLL	
-14	Preparing the .DLL and Import Library	
-15	Preparing the .EXE	
-16	Methods of Dynamic Linking	
-17	Dynamic Linking at Run Time	
-18	DosLoadModule	C.P. 2-151
-19	DosQueryProcAddr, DosFreeModule	C.P. 2-250, 117
-20	Data Objects in DLL's	
-21	DLL Initialization	
-22	Flow of Control for DLL Initialization	
-23	_DLL_InitTerm	U.G. 225-237
-24	Types of DLL Initialization	
-25	Initializing the C Run-time Environment	U.G. 234
-26	Invoking _CRT_init: App. Statically linked (/Gd-), No User DLLs	
-27	Invoking _CRT_init: App. Dynamically Linked (/Gd), No User DLLs	
-28	Invoking _CRT_init: App. and User DLL Dynamically Linked (/Gd)	
-29	Invoking _CRT_init: App. Dynamically Linked (/Gd), User DLL Static (/Gd-)	
17.	Exception Management	P.G.1 Chp. 13
-1	Introduction	
-2	System Exceptions	P.G.1 13-1
-3	Type of Exceptions	P.G.1 13-1
-4	Signal Exceptions	P.G.1 13-6
-5	Raising Exceptions	P.G.1 13-8
-6	Error Pop-Up Screens	P.G.1 13-13
-7	Exception Handlers	P.G.1 13-2,3,9
-8	Exception Management Structures	P.G.1 13-15
-9	ExceptionRegistrationRecord	P.G.1 13-16
-10	ExceptionReportRecord	P.G.1 13-15
-11	ContextRecord	P.G.1 13-17
-12	Exception Handler Example	
-13	Exception Handler Example cont'd	
-14	Summary of Exception Related Functions	
18.	Session Management	P.G.1 7-32 thru 37
-1	Introduction	
-2	Overview of Sessions, Processes and Threads	
-3	Session Resources	
-4	Session	
-5	Session Management Components	
-6	Session Hierarchy	
-7	Creating A Session	
-8	DosStartSession	C.P. 2-343
-9	Setting Selectability & Bonding of Child Session	P.G.1 7-34,35
-10	DosSetSession	C.P. 2-333

- 11 DosSelectSession
- 12 DosStopSession

C.P. 2-287
C.P. 2-353

19. Pipes

P.G.1 Chp. 9

- 1 Introduction
- 2 Standard Input/Output
- 3 Anonymous Pipes
- 4 Using Anonymous Pipes
- 5 DosCreatePipe, DosDupHandle
- 6 Pipes/Redirection Example
- 7 Named Pipes
- 8 Server/Client Communication Using Named Pipes
- 9 Named Pipe Server Program Example
- 10 Named Pipe Server Program Example cont'd
- 11 Named Pipe Client Program Example
- 12 Summary of Named Pipe Functions

C.P. 2-49, 72

P.G.1 9-2

P.G.1 9-3 thru 5

Index

Bibliography

Laboratory Exercises


Sample Program Listings

Introduction to OS/2

Design Goals of OS/2

Application Programming Interface

Multitasking Hierarchy

- 
- SESSIONS
 - PROCESSES
 - THREADS



Design Goals

Multitasking With Performance/Response Equal to Single Tasking:

- ✓ Parallel multitasking through preemptive scheduling using a single CPU.
- ✓ At any time, CPU usage may be unpredictably switched to another task.

Device Independence:

- ✓ The application issues commands to a generic device.
- ✓ The device driver translates the commands to fit the characteristics of the actual hardware.
- ✓ The application does not need to know the specifics of the hardware.



Design Goals

Custom and Stable Execution Environment:

- ① Extended physical memory addressing to 4 GB.
- ① Virtual memory addressing to 64 TB.
- ① Memory protection ensures errors in one application do not corrupt any other application.
- ① Isolation of applications from one another and from the operating system.

20NTR003 920708



Design Goals

- ✓ Fully exploit the capabilities of the Intel 80386 / 80486 processors.
- ✓ Provide a 32-bit programming environment that is portable to other architectures.
- ✓ Binary compatibility with the OS/2 1.x 16-bit executables.



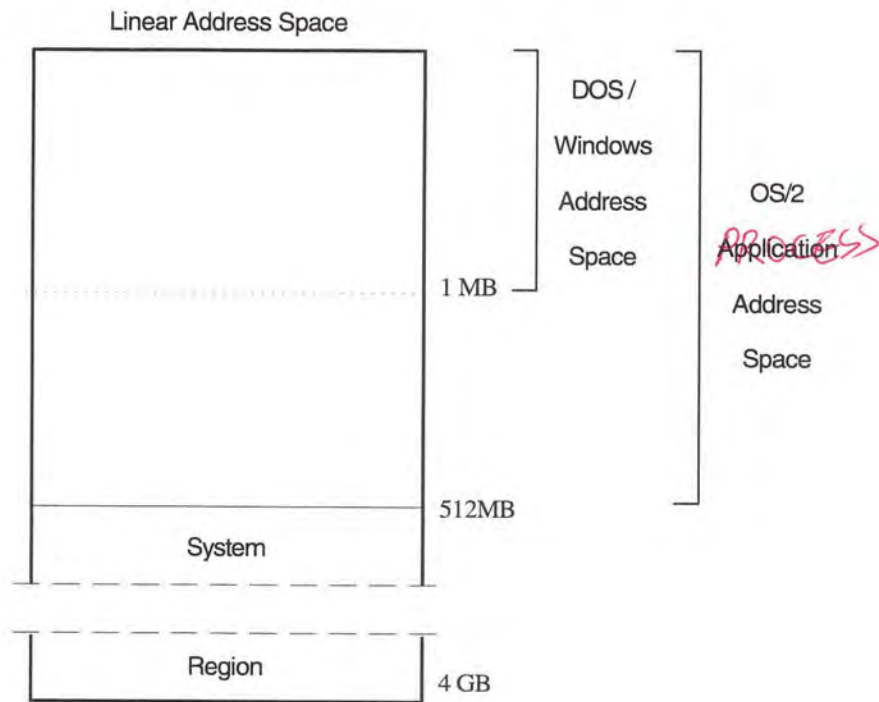
Design Goals

DOS / Windows Application Support In A Protected Environment:

- ✓ Up to 240 concurrent sessions.
- ✓ Over 630 KB of standard memory available to the application.
- ✓ Up to 32 MB of EMS, 16 MB of XMS and 512 MB of DPMI (DOS Protected Mode Interface) memory available to the application.
- ✓ Foreground or background execution in either a full screen or windowed session.

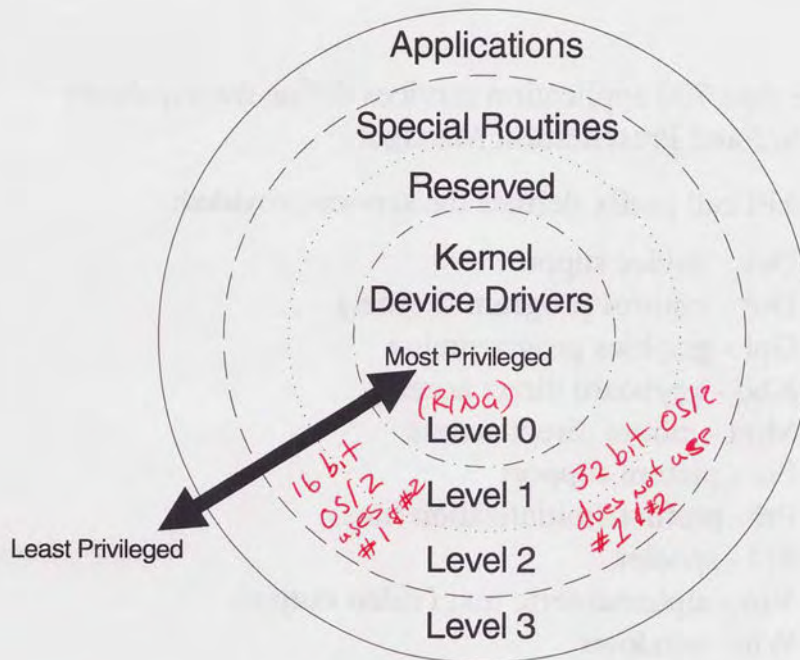
20NTR005 920708

OS/2 2.x Memory Organization



20VTR006 930910

OS/2 Privilege Model

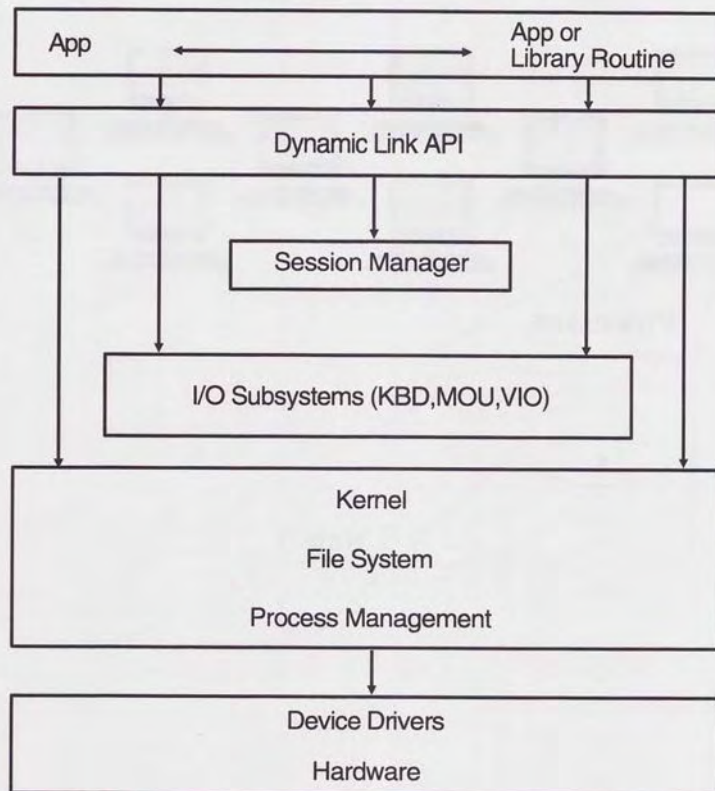


20NTR007 920708

Application Programming Interface (API)

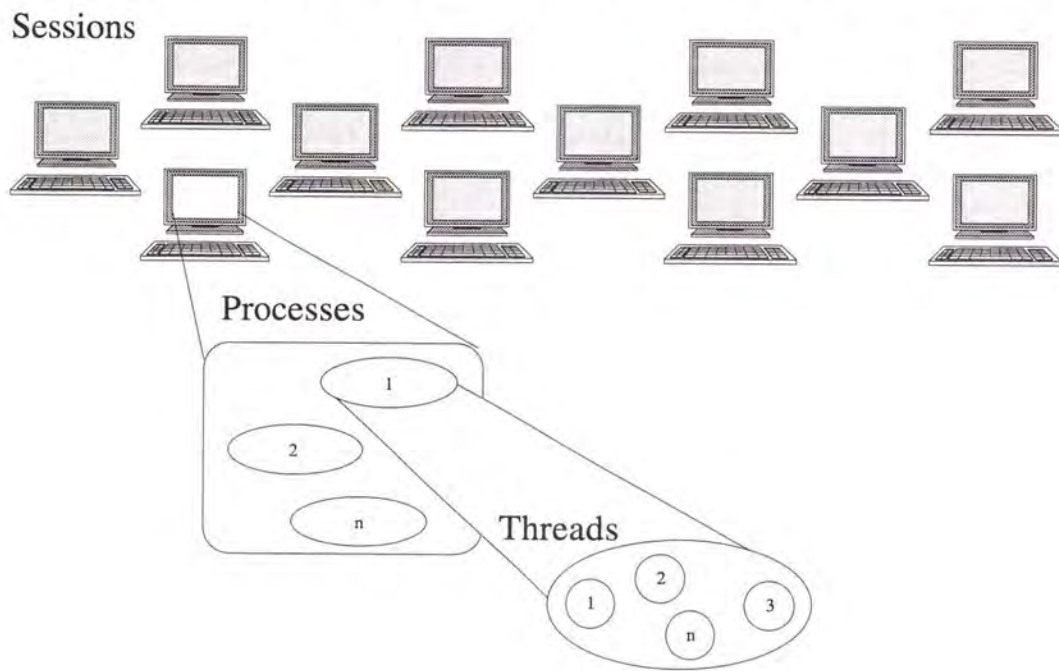
- ✓ More than 700 application services define the capability of OS/2 and Presentation Manager.
- ✓ The API call prefix defines the service provided:
 - Dev - device support
 - Dos - control program (kernel)
 - Gpi - graphics programming
 - Kbd - keyboard direct access
 - Mou - mouse direct access
 - Pic - picture support
 - Prf - profiles (initialization files)
 - Spl - spooler
 - Vio - alphanumeric text (video output)
 - Win - windows

Layers of OS/2



20NTR009 920708

OS/2 Multitasking Hierarchy



20NTRO10 920708

OS/2 Applications

OS/2 Mode Applications

DOS Mode Applications

20APPS01 920709

Types of Applications

✓ OS/2 Mode Windowed Applications:

Each applications runs in a separate session

May share the display with other windowed applications.

Applications multitask.

✓ OS/2 Mode Full-Screen Applications:

Applications do not share the display with other applications.

Applications multitask.

✓ DOS / Window Mode Applications:

OS/2 emulates PC-DOS.

MOST applications will run the same as under PC-DOS.

Applications may execute in a window or full screen.

Applications multitask.

OS/2 Mode Windowed Applications

✓ Presentation Manager Applications:

Creates one or more windows to present functions to the user.

May generate full graphics.

Should adhere to the SAA Common User Access definition.

Is aware that the system is being shared with other applications.

Applications multitask.

✓ Text-Windowed Applications:

Do not explicitly create windows.

OS/2 provides a default window for execution.

Applications multitask.

02APPS03 920709

OS/2 Mode Full-Screen Applications

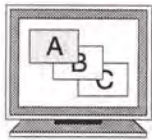
- ④ Do not share the display with other applications.
- ④ Each application runs in its own full-screen session.
- ④ Applications multitask.

DOS / Window Mode Applications

- ✓ Applications originally developed for PC-DOS that adhere to the PC-DOS specification.
- ✓ Window applications written to the 3.1 specifications.
- ✓ Applications execute in a window or full screen.

20APPS05 931101

Concurrent Execution of Application Classes



A

B

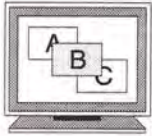
C

D

E

Foreground

Background



A

B

C

D

E

Foreground

Background



A

B

C

D

E

Foreground

Background



A

B

C

D

E

Foreground

Background



A

B

C

D

E

Foreground

Background

A PM app

B DOS/Window app

C Text-window app

D Full-screen app

E DOS/Window app

20APPS06 920709

System Limits - 2.x vs 1.3

	32 bit 2.0		16 bit 1.3	
	Sys	Usr	Sys	Usr
Sessions				
Full-Screen	255	255	16	12
Graphic-Window (PM)	255	252	16	14
Text-Window	255	252	16	16
MVDM (DOS)	255	252	1	1
Processes	4096	4065	511	504
Threads	4095	4065	512	483
Threads per Process	4095	4065	54	54
Open File Handles - system wide	64K		64K	
Global Semaphores - system wide	64000	64000	512	512
User signals per process			3	
Minimum stack for Thread		8K		4K
Timers per Thread			1	
Process Address Space		512M		512M

20APPS07 930910

Notes

Protected Mode

Register Set

Operating Modes

Accessing Memory

Exceptions

20PMOD01 920709

General Purpose Registers

			31	16	15	8	7	0	
General Purpose Registers	Accumulator	EAX				AH		AL	AX
	Base	EBX				BH		BL	BX
	Count	ECX				CH		CL	CX
	Data I/O	EDX				DH		DL	DX
	Source Index	ESI				SI			
	Destination Index	EDI				DI			
	Base Pointer	EBP				BP			
	Stack Pointer	ESP				SP			
Special Purpose	Instruction Pointer	EIP				IP			
		EFLAGS				FLAGS			

☐ Available in the 80386/486

20PMOD02 920709

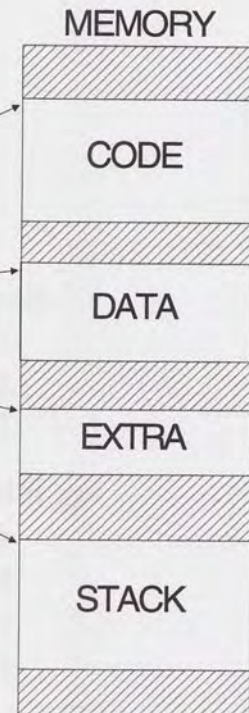
— error Codes Return to the Accumulator (EAX)

Memory Segment Registers

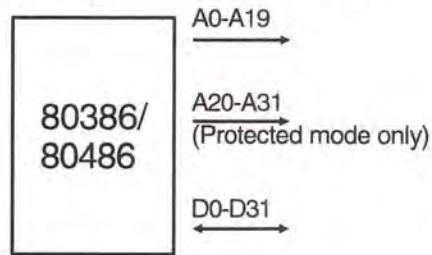
- ✓ The 8088/80286 have 4 Segment Registers:

CODE SEGMENT	(CS)
DATA SEGMENT	(DS)
EXTRA SEGMENT	(ES)
STACK SEGMENT	(SS)

- ✓ These Segment Registers point to the beginning of the 4 currently addressable memory segments.
- ✓ Additional segments registers, FS and GS, are available in the 80386/486.



Operating Modes



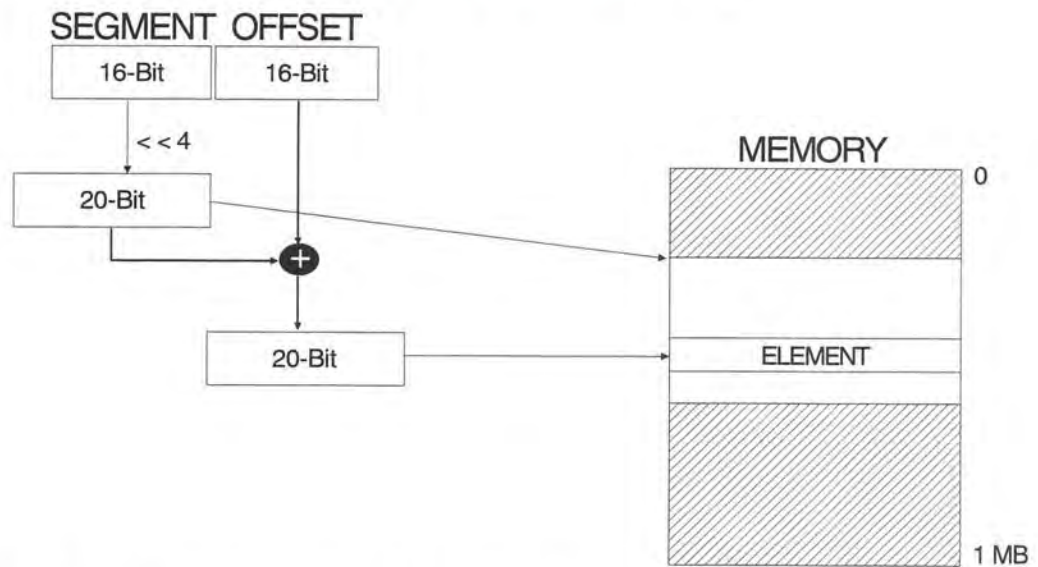
- ✓ Real Mode:
 - 1MB physical address space
 - Runs like a fast 8086
 - Default power-on mode
- ✓ Protected Mode:
 - 4 GB physical memory
 - Memory Protection
 - Hardware Task Switches
 - 64 Terabyte Per Task Virtual Memory Support
- ✓ Virtual 8086 Protected Mode
 - Supports execution of 8086/8088 programs in a protected mode environment.

20PMOD04 920709

Protection Mechanisms

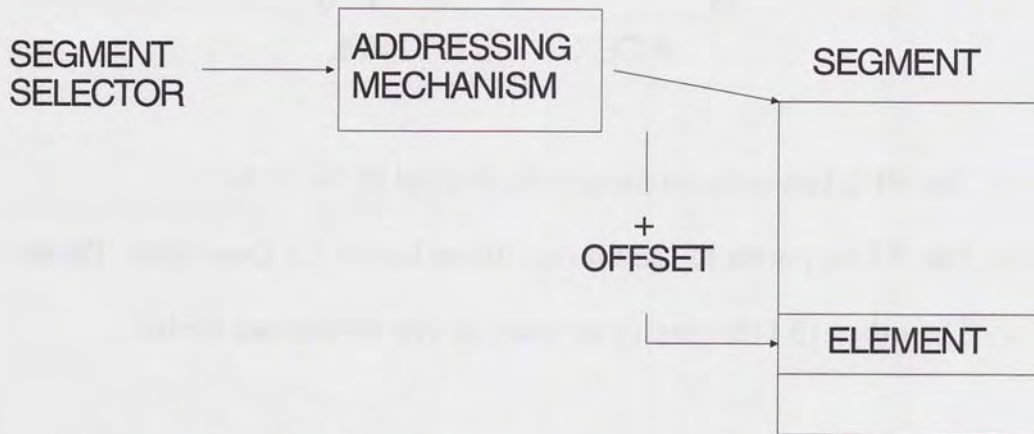
- ④ Protects the user from programming errors:
 - Out of range accesses
 - Incorrect I/O
 - Overlaying interrupt vectors
- ④ Protects processes from other processes:
 - Processes are isolated
- ④ Protects the operating system from the user:
 - Multiple privilege levels

Real Mode Memory Addressing



- ✓ There is a direct relationship between the segment register value and the physical memory address.
- ✓ $[\text{SEGMENT} * 16] + \text{OFFSET} = 20 \text{ bit physical address}$

Protected Mode Memory Addressing



- ✓ The address = Selector : Offset.
- ✓ The Selector does not point directly to memory.
- ✓ Selector : Offset can refer to an element not currently present in physical memory (logical address).

Selector Format

15		3	2	1	0
INDEX			TI	RPL	

- ✓ The RPL bits indicate the privilege level of the task.
- ✓ The TI bit points to the Global (0) or Local (1) Descriptor Table.
- ✓ The other 13 bits specify an entry in one of the two tables.

$$2^{32} \text{ BYTES/SEGMENT} \times 2^{14} \text{ SEGMENTS} = 2^{46} \text{ BYTES}$$

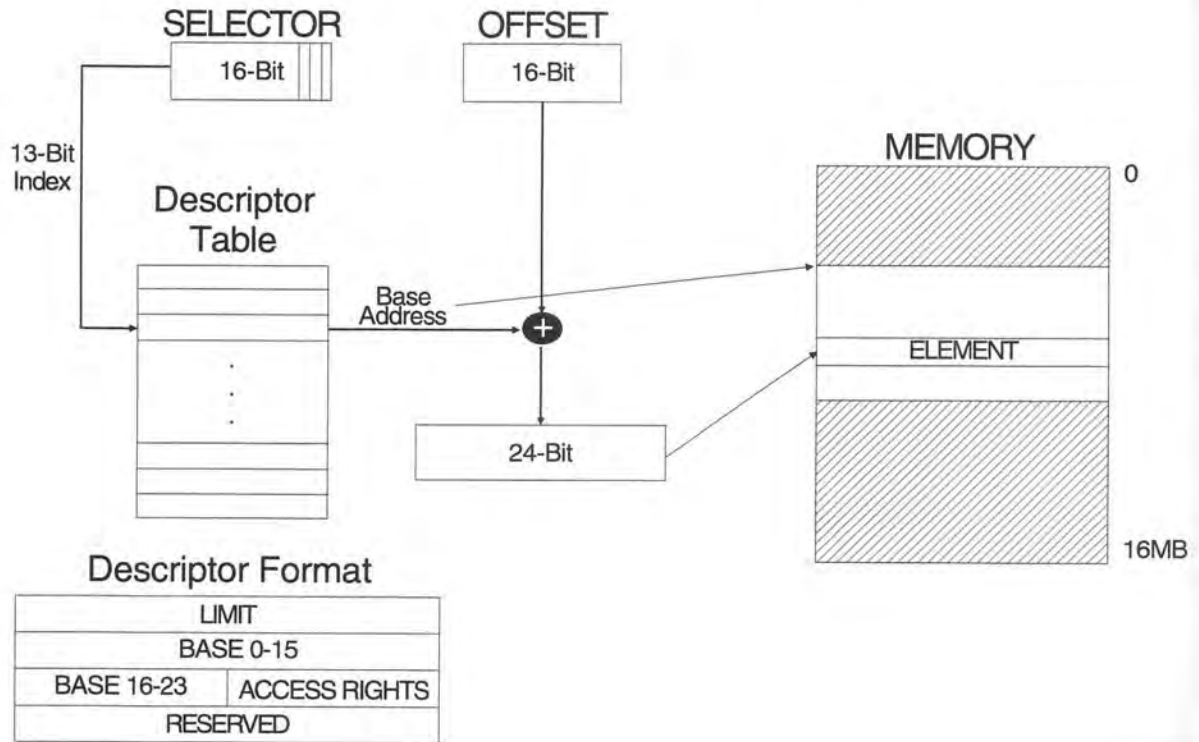
Data Segment Descriptors

LIMIT							
BASE 0-15							
BASE 16-23				ACCESS RIGHTS			
G	B	O	A V L	LIMIT 16-19	BASE 24-31		

☐ Available in 80386/486. Reserved in the 80286

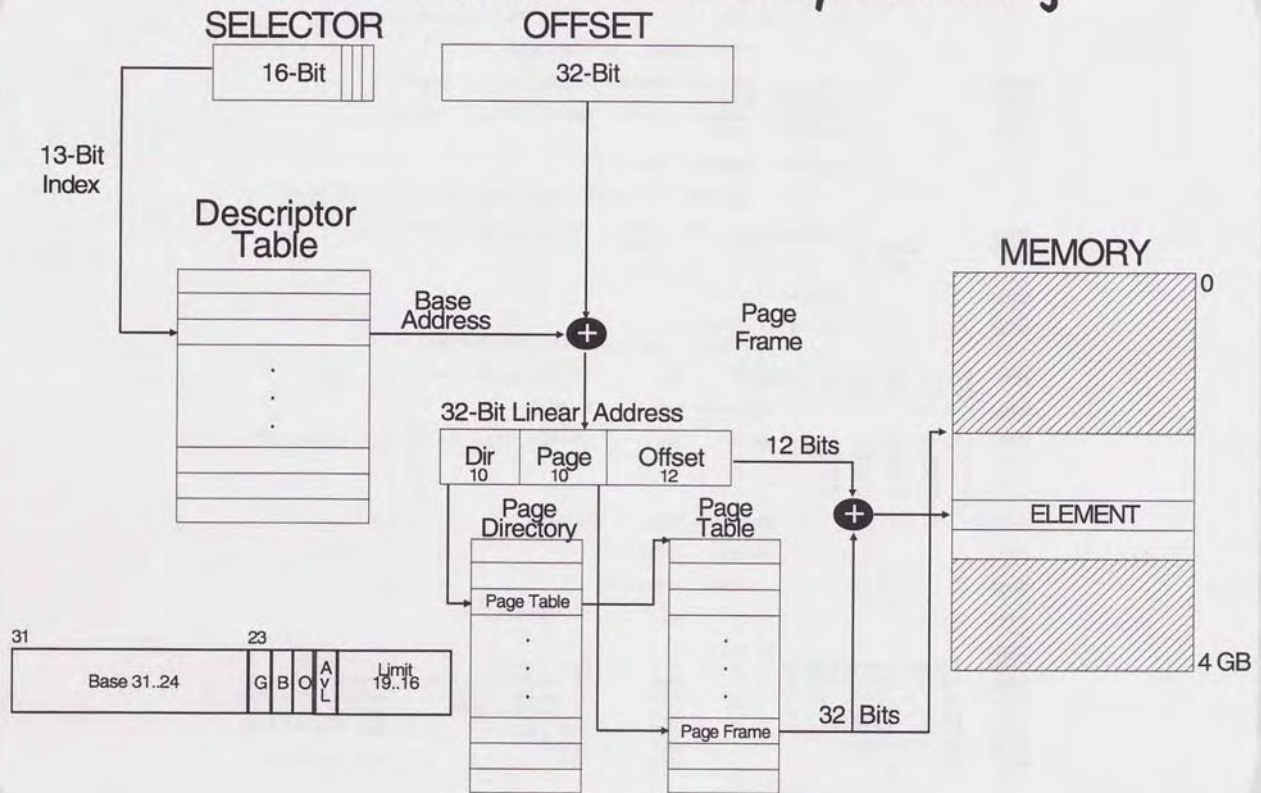
- ✓ 8 bytes that describe a Data Segment.
- ✓ Contains a full 32-bit address (24-bit in the 80286).
- ✓ Contains segment size (limit) and access rights.

16-Bit Protected Mode Memory Addressing



20PMOD09 920709

32-Bit Protected Mode Memory Addressing



20PMOD10 930719

FAULT.LST

```
.386                ; allow 386 instructions
INCLUDELIB OS2386.LIB

0000 CODE32 SEGMENT DWORD USE32 PUBLIC 'CODE'
0000 CODE32 ENDS
0000 DATA32 SEGMENT DWORD USE32 PUBLIC 'DATA'
0000 DATA32 ENDS

DGROUP GROUP data32

ASSUME CS:FLAT, DS:FLAT, SS:FLAT, ES:FLAT

0000 stack32 SEGMENT DWORD USE32 STACK 'STACK'
0000 db 8192 DUP (?)

2000 2000 [
0000 00 l
1 stack32 ENDS

0000 data32 SEGMENT
PUBLIC msg1,msgllen,msg2,msg2len

0000 41 20 6D 65 73 73 msg1 db 'A message',0dh,0ah
0000 61 67 65 0D 0A
000B = 000B msgllen equ $-msg1

000B 48 65 6C 6C 6F 20 msg2 db 'Hello OS/2 World !!!',0dh,0ah
000B 4F 53 2F 32 20
000B 57 6F 72 6C 64
000B 20 21 21 21 0D
0A
0021 = 0016 msg2len equ $-msg2

0021 data32 ENDS

0000 code32 SEGMENT

0000 77777777 var1 dd 77777777h

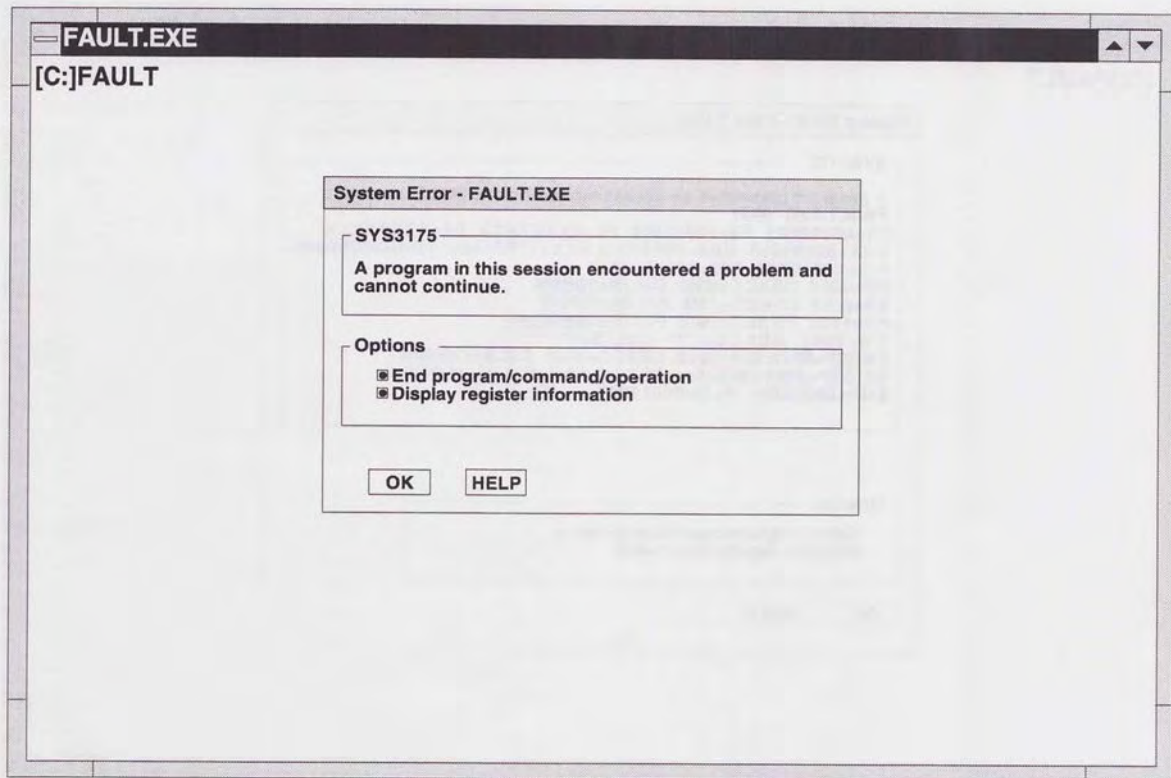
0004 Start:
0004 8B 0D 00000000 R mov ecx,var1 ; access OK
000A 8A 2D 0000000B R mov ch,msg1+msgllen ; access OK
0010 8A 0D 00000021 R mov cl,msg2+msg2len ; access OK. WHY ??
0016 8A 0D 00001000 R mov cl,msg2+0ff5h ; 1st violation
001C 89 0D 00000000 R mov var1,ecx ; 2nd violation
0022 BC 00000000 mov esp,0
0027 50 push eax ; 3rd violation

; 4th violation

0028 code32 ENDS
END Start
```

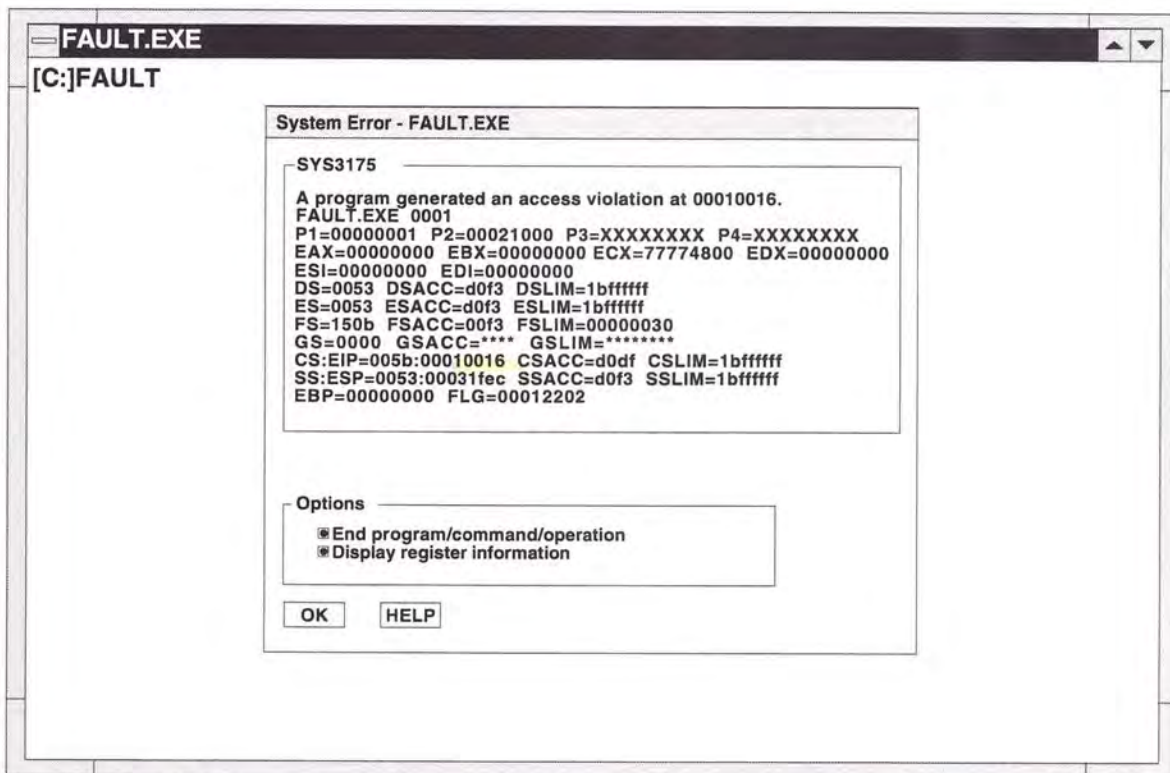
20PMOD18 920713

First Violation - Part 1



20PMOD12 920713

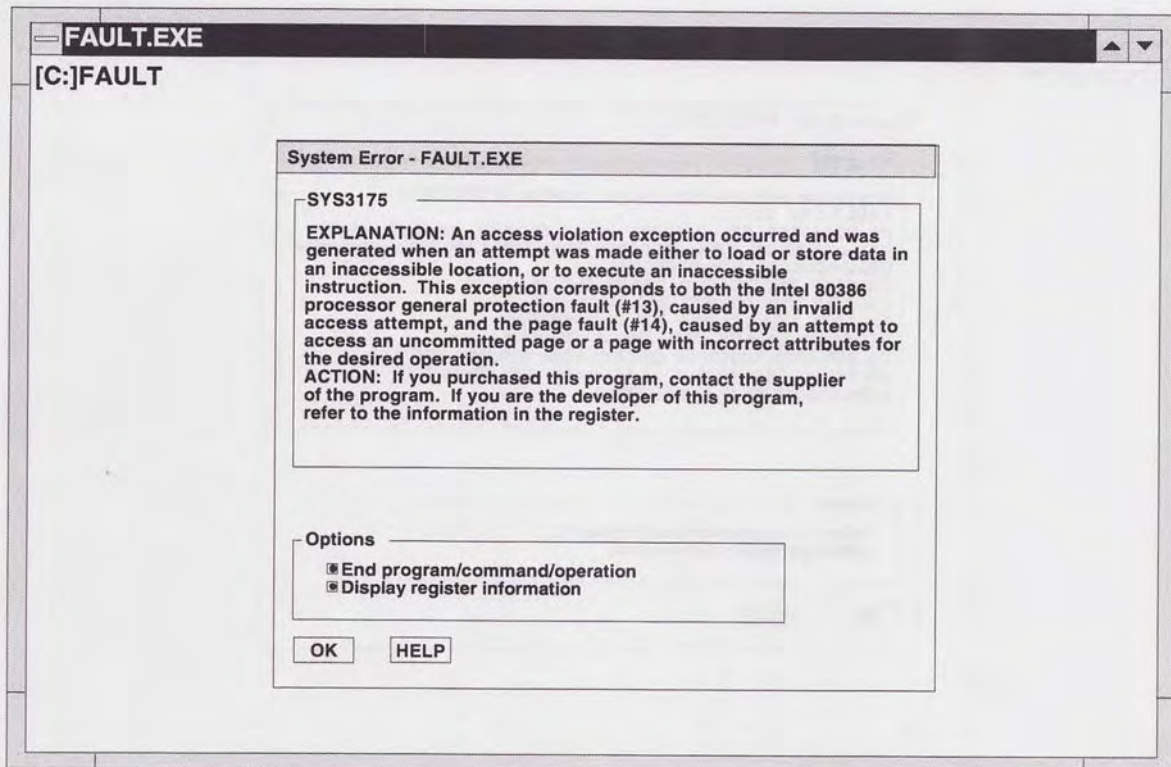
First Violation - Part 2



20PMOD13 920713

- highlighted is virtual address fault happened
(10016)

First Violation - Part 3



20PMOD14 920713

Second Violation



20PMOD15 920713

Third Violation



20PMOD16 920713

Fourth Violation



20PMOD17 920713

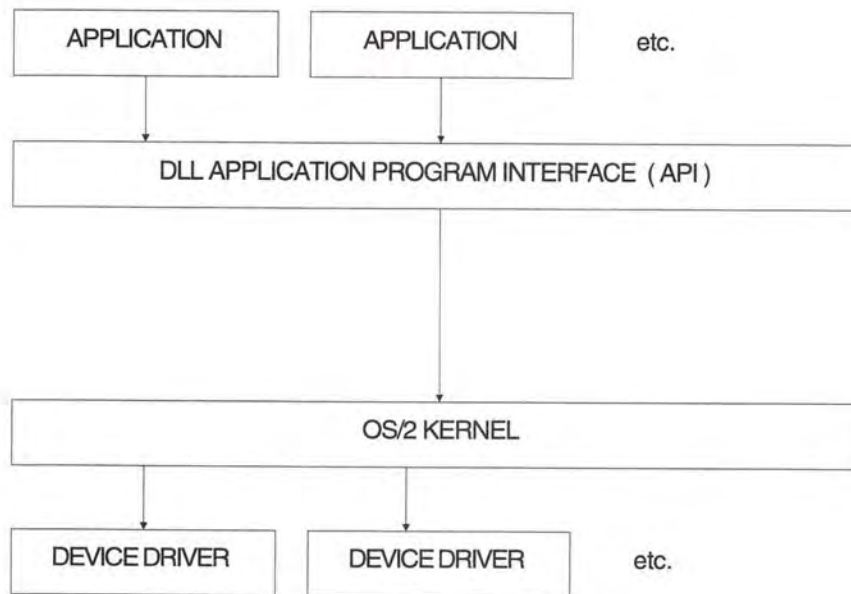
Introduction to OS/2 Programming

API interface
Return Codes

20PRGM01 920710

DISK 1 — COURSE MATERIALS
DISK 2 — INP FILES (EXTRAS)

OS/2 Application Interface




20PRGM02 920710

DOS Register-Based API

APPLICATION

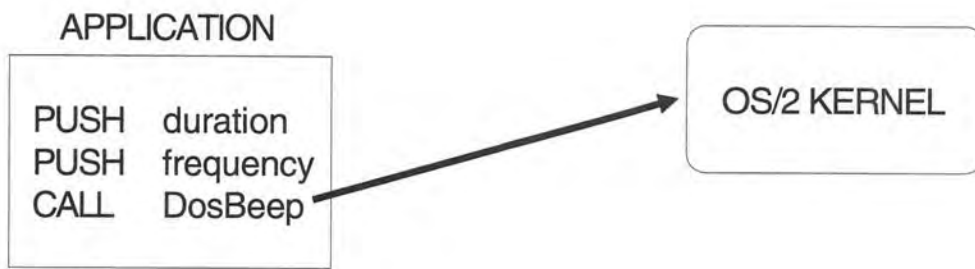
```
; terminate program  
MOV  AH, 4Ch  
INT  21H
```



PC-DOS

- ✓ Pass parameters via registers.
- ✓ Link to DOS through a software interrupt.
- ✓ Very efficient but, difficult to do from a high-level language.

OS/2 Call-Based API



- ✓ Passing parameters on the stack is compatible with most high level languages.

Using the C Language for OS/2

```
#define INCL_BASE
#define INCL_NOPMAPI
#include <os2.h>

int main ()
{

    return (NO_ERROR);
}
```

STANDARD LOCATION of .h files
SET UP IN CONFIG.SYS
VAR = INCLUDE

- ✓ OS2.H contains the API function prototypes.
- ✓ INCL_BASE limits the processing of function prototypes to base kernel functions.

API Function Prototypes

→ causes stack
passing of
params

```
VOID APIENTRY DosExit ( ULONG action, /* 0=end current thread, 1=end process */  
                        ULONG result); /* program completion code*/
```

```
APIRET APIENTRY DosBeep ( ULONG freq, /* frequency of sound in hertz */  
                          ULONG dur); /* length of sound in milliseconds */
```

- ✓ ANSI 'C' Standard
- ✓ Extended 'C' compiler error checking
- ✓ Interface is independent of model of compilation used

20PRGIM06 921020

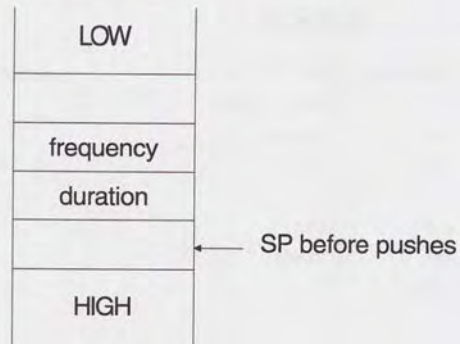
→ HELP compiler
command line METHOD

> View PROGRAM21 DOSEXIT

System Calling Convention

APIRET _System DosBeep (ULONG, ULONG);

DosBeep (frequency , duration);



- ✓ Parameters are pushed on the stack in right-to-left order as in normal C.
- ✓ The 'caller' clears parameters from the stack.
- ✓ The IBM C/C++ compiler's default method of passing parameters is to use a combination hardware registers and the stack. This is called Optlink.
- ✓ OS/2 ver. 1x uses the pascal calling convention .

20PRGM07 930910

Return Codes

MASM

```
@DosBeep ( 50, 300 );  
OR      EAX , EAX  
JNZ     error  
(JZ)    or  
  
CMP     EAX,0  
JNE     error  
(JE)
```

C

```
APIRET rc;  
  
rc = DosBeep (50, 300);  
if (rc) printf ( "rc=%d Unable to beep speaker\n",rc);
```

- ✓ Always returned in the EAX register. 0 = NO_ERROR.
- ✓ Return codes are documented in the OS/2 2.0 Technical Library reference materials.

20PRGM08 930910

*be
baser.h
error Message file*

Class Exercise

Given the following 'C' function prototype, how many bytes of stack space are necessary for the parameters?

```
APIRET WINAPI DosWrite (  
    HFILE hFile,           // File handle  
    PVOID pBuffer,        // Output buffer  
    ULONG cbWrite,         // Number of bytes to be written  
    PULONG pcbActual);     // Number of bytes written (returned)
```

20PRGM09 920710

Notes

Environment Strings and Command Line Arguments

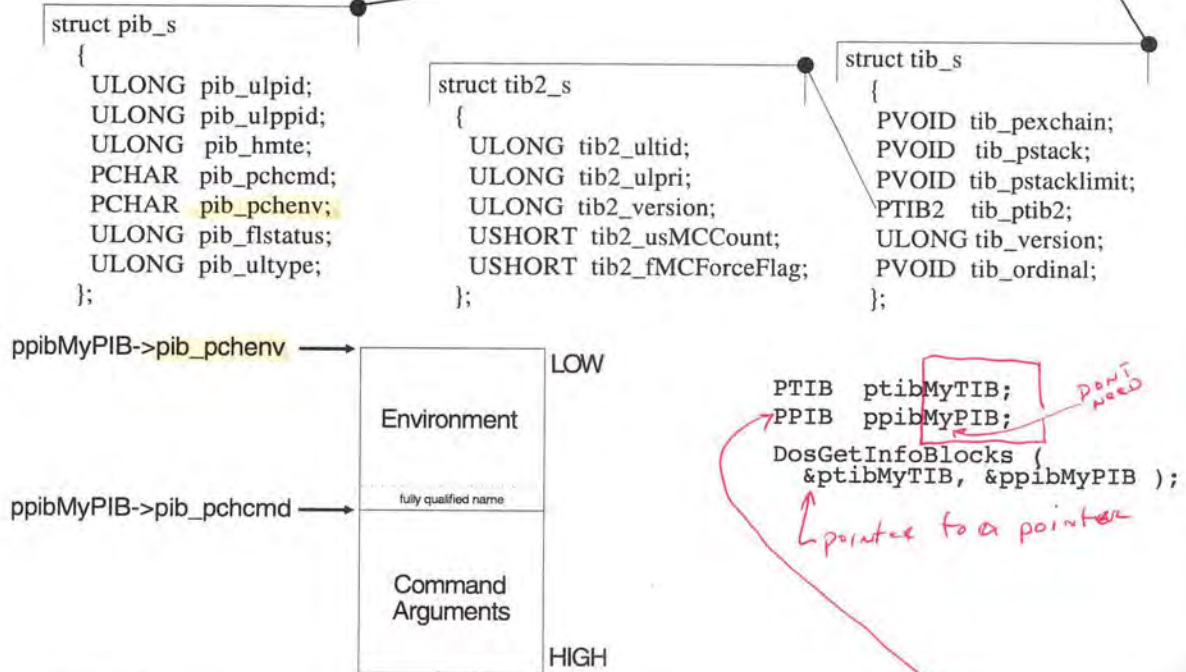
Accessing the Environment

Accessing the Command Line

20ENVR01 920710

Obtaining Environment and Command Line Pointers

DosGetInfoBlocks (PTIB *ptib, --> ptr to ThreadInfoBlock1
PPIB *ppib); --> ptr to ProcessInfoBlock



- See Vol 2 page Lab1b-3 for structure of environment of PPIB

ESTABLISHES PTR TO ENVIRONMENT STRUCTURE

Searching the Environment

DosScanEnv (PSZ pszName, <-- ptr to Name string (in)
PSZ *ppszValue);--> ptr to variable to receive ptr to Value (out)

0 NO_ERROR
203 ERROR_ENVVAR_NOT_FOUND

- ✓ Searches the environment for a specified ASCII string (Name).
- ✓ The Name string must not include the equal (=) character.
- ✓ Returns a pointer to the value assigned to the environment variable.

```
PSZ pszName = "PATH";  
PSZ pszValue;  
  
DosScanEnv ( pszName, &pszValue );  
printf ("Path = %s\n", pszValue);
```

20ENVR03 921008

Environment Strings

- ✓ Each process owns a set of environment strings.
- ✓ Each string has the form: <name>=<value> \0.
- ✓ The last string is terminated with \0\0.
- ✓ The fully expanded path of the executable file follows the last environment string.
- ✓ Example:

icc /c /Ss /Ti /W3 /Gs browse.c

30000	43 4F 4D 53 50 45 43 3D 43 3A 5C 43 4D 44 2E 45	COMSPEC=C:\CMD.E
30010	58 45 00 50 41 54 48 3D 43 3A 5C 3B 43 3A 5C 49	XE.PATH=C:\;C:\I
30020	42 4D 43 32 5C 42 49 4E 3B 00 44 50 41 54 48 3D	BMC2\BIN;.DPATH=
30030	43 3A 5C 3B 43 3A 5C 49 42 4D 43 5C 42 49 4E 3B	C:\;C:\IBMC\BIN;
30040	20 00 00 43 3A 5C 49 42 4D 43 32 5C 42 49 4E 49	.C:\IBMC\BIN\I
30050	43 43 2E 45 58 45 00	CC.EXE.

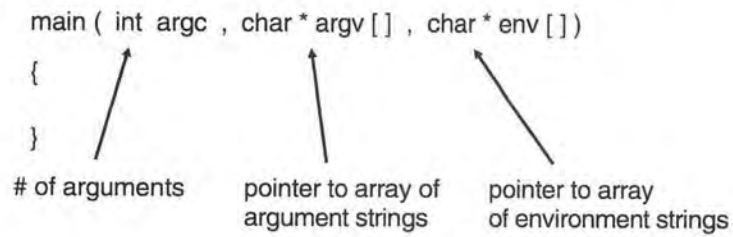
Command Line

- ✓ Unlike the C compiler, OS/2 does not format the command line.
- ✓ The command (program name) is terminated with \0
- ✓ The the remainder of the command line is terminated with \0\0
- ✓ Example:

C> icc /c /Ss /Ti /W3 /Gs browse.c

```
30057 69 63 63 00 20 2F 63 20 2F 53 73 20 2F 54 59 20 icc. /c /Ss /Ti
30067 2F 57 33 20 2F 47 73 20 42 52 4F 57 53 45 2E 43 /W3 /Gs browse.c
30077 00 00 ..
```


'C' Program Entry State



✓ 'C' startup code formats the arguments into null-terminated strings.

✓ EXAMPLE:

```
C>environ this is a test <cr>
```

```
argc = 5  
argv[0] = "environ"  
argv[1] = "this"  
argv[2] = "is"  
argv[3] = "a"  
argv[4] = "test"
```

DosScanEnv Program Example

```
// scanenv.c -- displays the current path
***** OS/2 Version 2.x *****

#define INCL_BASE
#include <os2.h>
#include <stdio.h>

#define STDOUT 1

// internal function prototypes
int terminate ( PSZ pszMsg, APIRET rc, PSZ pszMyName );
// Globals

int main (void)
{
    APIRET rc;
    PSZ pszMyName = "SCANENV.";
    PSZ pszName = "PATH";
    PSZ pszValue;

    rc = DosScanEnv ( pszName, &pszValue );
    if (rc) terminate ("ScanEnv error !\n", rc, pszMyName);
    printf ("The current path is - %s\n", pszValue);
    return ( NO_ERROR );
}

/*****
int terminate ( PSZ pszMsg, APIRET rc, PSZ pszMyName)
{
    CHAR chMsg[255];
    PSZ pszMsgFile = "OSO001.MSG";
    ULONG ulMsgLen;

    printf ("%s: Error #%d %s \n", pszMyName, rc, pszMsg);
    if (rc != 0 & rc != -1)
    {
        DosGetMessage (0, 0, (PCHAR) chMsg, 255L, rc, pszMsgFile, &ulMsgLen);
        DosPutMessage (STDOUT, ulMsgLen, chMsg );
    }
    exit (1);
}
*****/
```

20ENV07 931109

Notes

Program Development

Programming Tools
Development Process

20DLVP01 920711

Programming Tools

- ✓ Include and Header files
- ✓ Sample Programs
- ✓ Presentation Manager Utilities
- ✓ Help Manager Utilities
- ✓ Message Utilities
- ✓ WorkFrame/2

20DLVP02 920711

Header and Include Files

- ✓ The INCLUDE statement automatically calls a hierarchy of header/include files.
- ✓ The DEFINE/EQU statements are used to control which group of functions are included.
- ✓ Header/include file hierarchy:

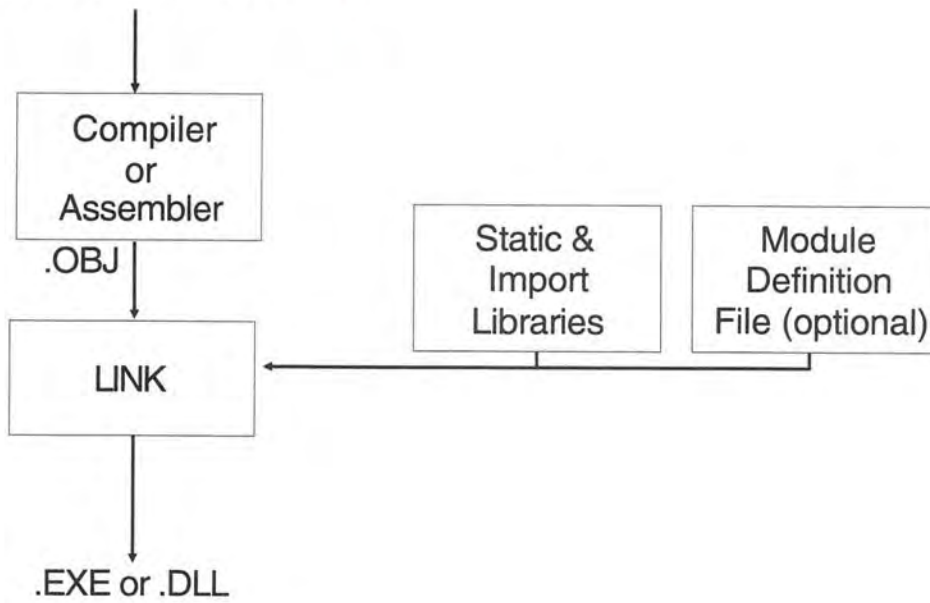
OS2.H (.INC)	PM	
OS2DEF	PMWIN	PMERR
BSE	PMGPI	PMHELP
BSEDOS	PMDEV	PMMLE
BSESUB	PMAVIO	PMSEI
BSEERR	PMSPL	PMSHL
BSEDEV	PMPIC	PMWP
BSEMEMF	PMORD	
BSEORD	PMBITMAP	
BSETIB	PMFONT	
BSEXCPT	PMDDI	
	PMDDIM	

20DLVP03 920711

*TB =
Thread Information
Block*

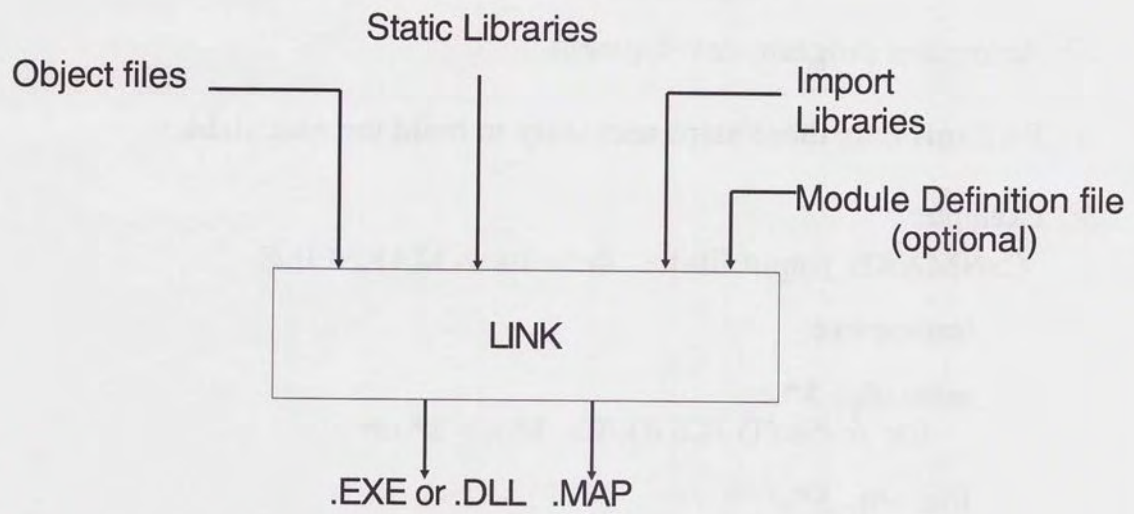
OS/2 Development Process

Source and Include files



20DLVP04 920711

OS/2 Linker



20DLVP06 920711

NMAKE

Program Maintenance Utility

- ✓ Automates program development.
- ✓ Performs only those steps necessary to build the executables.
- ✓ Example:

C>NMAKE [input file] defaults to MAKEFILE

browse.exe:

main.obj: *.c

icc /c /Ss /Ti /Kb /Q /Rn *.c > *.err

tool.obj: *.c

icc /c /Ss /Ti /Kb /Q /Rn *.c > *.err

screen.obj: *.c

icc /c /Ss /Ti /Kb /Q /Rn *.c > *.err

browse.exe: main.obj tool.obj screen.obj main.def

LINK386 /DE /LI /BASE:0x10000 main tool screen, , , main;

20DLVP05 930910

Memory Management

OS/2 Memory Model

Memory Objects

Allocation of Memory Objects

Suballocation of Memory Objects

20MAP101 920726

- static Allocation - ex. program load
- dynamic Allocation - mem. alloc when pgm needs.
(^{4K min} DosAllocMem)
- automatic Allocation - variable declares (stack alloc.)
- use Malloc for small memory allocation.

→ Actually uses 64k memory boundary, each ~~alloc~~ ~~DosAllocMem~~
DosAllocMem gives 4k min, but uses 64k shared
memory!

The Flat Memory Model

- ✓ OS/2 2.x defines a 32-bit memory model that is designed to be portable to any 32-bit uniprocessor or multiprocessor architecture including RISC.
- ✓ The model provides for memory objects that are larger than 64KB or larger than physical memory.
- ✓ The 80386/486 processors provide for a paged-virtual memory and 32-bit wide segments which may be up to 4GB in size.
- ✓ The 32-bit segment can be used to simulate a large, flat 32-bit virtual address space that is contiguous.
- ✓ On the 80386/486, a segmented virtual address (16:16 or 16:32) is translated by descriptors into a 32-bit (flat) linear address, which in turn is mapped to a 32-bit physical address by page tables.
- ✓ A pair of code and data descriptors maps the entire linear address space, effectively 'flattening' the virtual address space.

Virtual Address Space

- ✓ In the flat model, a linear address is a virtual address. A virtual address is also called a 0:32 address.
- ✓ The System Virtual Address Space is mapped by a pair of GDT code and data descriptors with limit fields of 4GB.
- ✓ The Process Virtual Address Space is mapped by a pair of GDT code and data descriptors with limit fields of 512MB. The 512MB limit provides compatibility with 16-bit applications and may be removed in the future.
- ✓ The process is provided with an independent 512MB linear address space by its own set of page directories and page tables.
- ✓ The virtual address space is partitioned in to private and shared regions.
- ✓ Private memory is allocated from low addresses toward higher addresses and shared memory is allocated from high addresses toward lower addresses.

20MAP103 920726

Memory Objects

- ✓ The smallest memory unit in the flat model is a page (4KB).
- ✓ A memory object is a range of contiguous linear pages within the process virtual address space.
- ✓ All memory objects are addressable simultaneously.
- ✓ Memory objects are nonrelocatable and, the pages that compose the objects are swappable.

Types of Memory Objects

Application Code	Private Address, Shared Storage
Application R/O Data	Private Address, Shared Storage
Application R/W Data	Private Address, Private Storage
Application Allocated Private Memory	Private Address, Private Storage
Dynamic Link Allocated Private Memory	Private Address, Private Storage
Application Allocated Shared Memory	Shared Address, Shared Storage
Dynamic Link Allocated Shared Memory	Shared Address, Shared Storage
Dynamic Link Instance Data	Shared Address, Private Storage
Dynamic Link Static Data	Shared Address, Shared Storage
Dynamic Link Code	Shared Address, Shared Storage

20MAPI05 930910

DosAllocMem

DosAllocMem (PPVOID ppb, --> pointer to pointer
 ULONG cb, <-- size in bytes
 ULONG flag); <-- allocation flags

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT

PAG_READ	0x00000001
PAG_WRITE	0x00000002
PAG_EXECUTE	0x00000004
PAG_COMMIT	0x00000010
OBJ_TILE	0x00000040
fPERM (PAG_EXECUTE+PAG_READ+PAG_WRITE)	
fALLOC (OBJ_TILE+PAG_COMMIT+fPERM)	

```
PVOID pArray6;
```

```
ulAllocFlags=PAG_READ | PAG_WRITE | PAG_COMMIT;  
DosAllocMem ( &pArray6, 0x200, ulAllocFlags );
```

20MAPI06 921020

- creates memory object who's minimum size is 4k
 and ~~used~~ uses a minimum of 64k ^(virtual) shared ~~space~~ space.

bse^(mf)~~error~~.h contains declare/prototype

DosFreeMem

DosFreeMem (PVOID pb); <-- base address of memory object

0	NO_ERROR
5	ERROR_ACCESS_DENIED
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

- ✓ Deallocates a private memory object.
- ✓ For a shared memory object, the reference count is decremented. If the resulting count is zero (0), the object is deallocated.

DosFreeMem (pb);

20MAP17 921020

- don't bother if your process is exiting. OS/2 will do it for you.

DosQueryMem

DosQueryMem (PVOID pb, <-- starting address
 PULONG pcb, <--> size of region
 PULONG pFlag); --> flags found

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

PAG_READ	0x00000001
PAG_WRITE	0x00000002
PAG_EXECUTE	0x00000004
PAG_GUARD	0x00000008
PAG_COMMIT	0x00000010
PAG_SHARED	0x00002000
PAG_FREE	0x00004000
PAG_BASE	0x00010000

- ✓ Returns attribute information about a range of pages within the process virtual address space.
- ✓ The query terminates at the end of the region or when a non-matching attribute is detected or when the base page of an object is encountered.

DosQueryMem (pb, &cb, &Flag);

Base 0x00480000	Length 0x00001000	Flags 0x00010017
Base 0x00481000	Length 0x0000F000	Flags 0x00000007
Base 0x00490000	Length 0x00001000	Flags 0x00010017
Base 0x00491000	Length 0x0000F000	Flags 0x00000007
Base 0x004A0000	Length 0x00001000	Flags 0x00010013
Base 0x004A1000	Length 0x0000F000	Flags 0x00000003

20MAPI08 921020

almost never used

DosSetMem

DosSetMem (PVOID pb, <-- region address
 ULONG cb, <-- number of bytes
 ULONG flag); <-- access requested

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
212	ERROR_LOCKED
487	ERROR_INVALID_ADDRESS
32798	ERROR_CROSSES_OBJECT_BOUNDARY

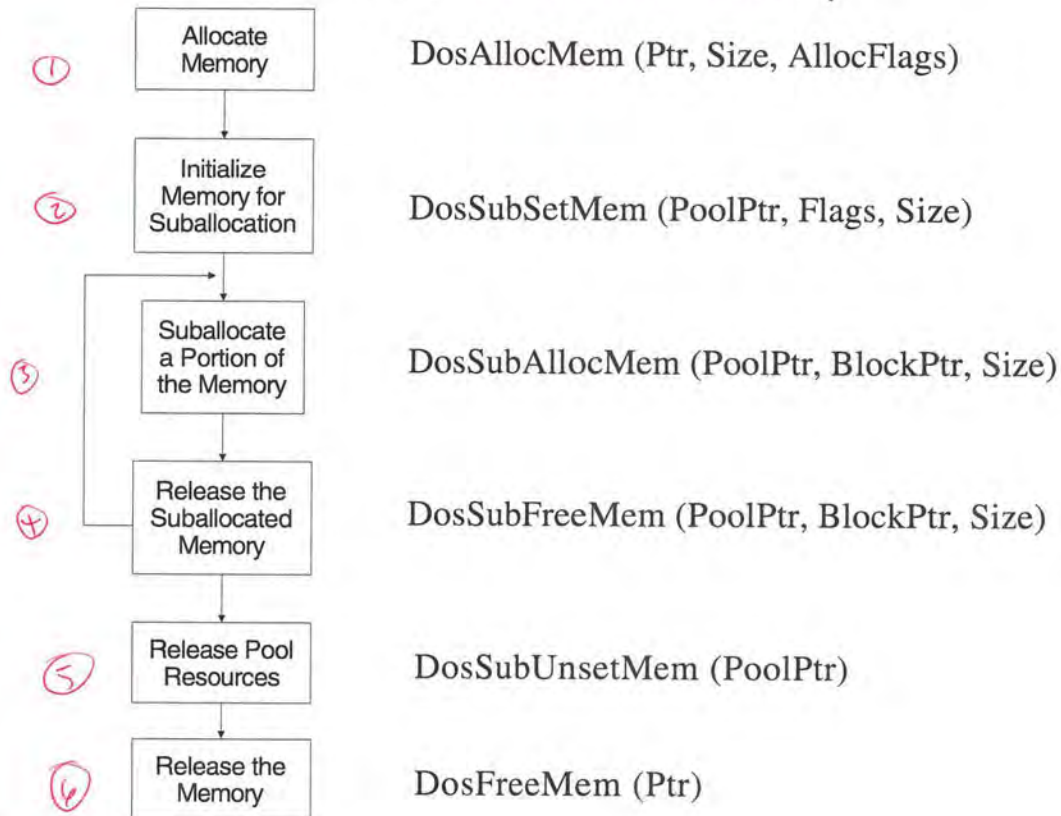
PAG_READ	0x00000001
PAG_WRITE	0x00000002
PAG_EXECUTE	0x00000004
PAG_GUARD	0x00000008
PAG_COMMIT	0x00000010
PAG_DECOMMIT	0x00000020
PAG_DEFAULT	0x00000400
fPERM (PAG_EXECUTE+PAG_READ+PAG_WRITE)	
fSET (PAG_COMMIT+PAG_DECOMMIT+PAG_DEFAULT+fPERM)	

✓ Sets the attributes of a range of pages within a memory object.

DosSetMem (pb, 4096, PAG_COMMIT | PAG_DEFAULT);

20MAPI 09 921020

Suballocation of Memory



OS/2 only
Will work between
processes, i.e. shared
memory.

DosSubSetMem

DosSubSetMem (PVOID pb,
 ULONG flag,
 ULONG cb);

<-- address of memory pool
<-- suballocated object characteristics
<-- size in bytes of the pool

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
310	ERROR_DOSSUB_SHRINK

DOSSUB_INIT	0x01
DOSSUB_GROW	0x02
DOSSUB_SPARSE_OBJ	0x04
DOSSUB_SERIALIZE	0x08

- ✓ Initializes, increases in size or enables access by another process to the memory pool. The first process does DOSSUB_INIT.
- ✓ The first 64 bytes are used by the pool manager.
- ✓ If suballocating between threads in the same process or another process, DOSSUB_SERIALIZE will ensure that accesses are serialized.
- ✓ If the 'parent' object was allocated sparse, DOSSUB_SPARSE_OBJ is necessary to have memory committed as necessary.

```
DosSubSetMem ( pb, DOSSUB_INIT | DOSSUB_SPARSE_OBJ, 16384 );
```

20MAPI11 940125

DosSubAllocMem, DosSubFreeMem

DosSubAllocMem (PVOID pbBase, <-- address of memory pool
 PPVOID ppb, --> address of block allocated
 ULONG cb); <-- size requested in bytes

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
311	ERROR_DOSSUB_NOMEM
532	ERROR_DOSSUB_CORRUPTED

```
PVOID pb;
DosSubAllocMem ( pbBase, &pb, 256 );
```

DosSubFreeMem (PVOID pbBase, <-- address of memory pool
 PVOID pb, <-- address of block to be freed
 ULONG cb); <-- size in bytes

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
312	ERROR_DOSSUB_OVERLAP
532	ERROR_DOSSUB_CORRUPTED

```
DosSubFreeMem ( pbBase, pb, 56 );
```

20MAP112 921020

DosSubUnsetMem

DosSubUnsetMem (PVOID pbBase); <-- address of memory pool

0	NO_ERROR
532	ERROR_DOSSUB_CORRUPTED

- ✓ Ends the use of a memory pool.
- ✓ Releases the resources used to manage the suballocation of the pool.
- ✓ Each thread that calls DosSubSetMem must call DosSubUnsetMem prior to the memory object being freed.

DosSubUnsetMem (pbBase);

20MAP113 921020

- do this before freeing memory.

Notes

Introduction to Multitasking

Serial Multitasking

Parallel Multitasking

OS/2 Elements of Multitasking

20MULT01 921026

Multiple Tasks

✓ Serial Tasks

- Read a manual
- Answer the phone
- Write a memo
- Create a spreadsheet
- Read electronic mail
- Schedule a meeting

✓ Parallel Tasks

- Drive a car and talk on the phone
- Jog and listen to the radio
- Print a chart, download a data file and format a disk

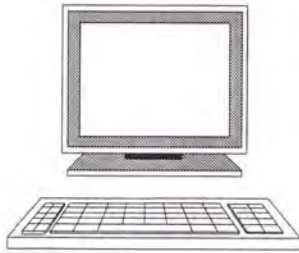
Parallel Multitasking

- ✓ True parallel multitasking requires multiple processors (CPUs).
- ✓ Currently, multiprocessor personal computers are rare.
- ✓ OS/2 is not considered a serial multitasking system.
- ✓ Preemptive scheduling of tasks gives the effect of parallelism.

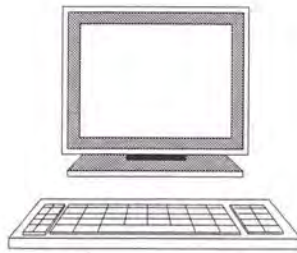
20MULT03 921026

OS/2 Elements of Multitasking

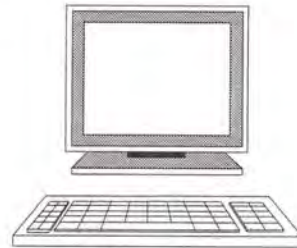
SESSIONS



Personal Information
Manager



Communication
Manager

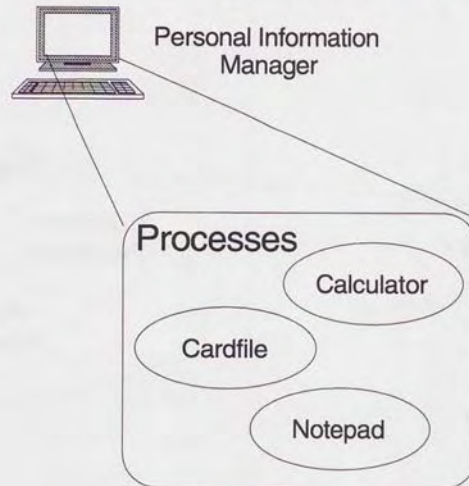


Database
Manager

- ✓ Each session is a virtual console device with its own keyboard video and mouse buffers.
- ✓ Switching between sessions usually means switching between applications.

OS/2 Elements of Multitasking

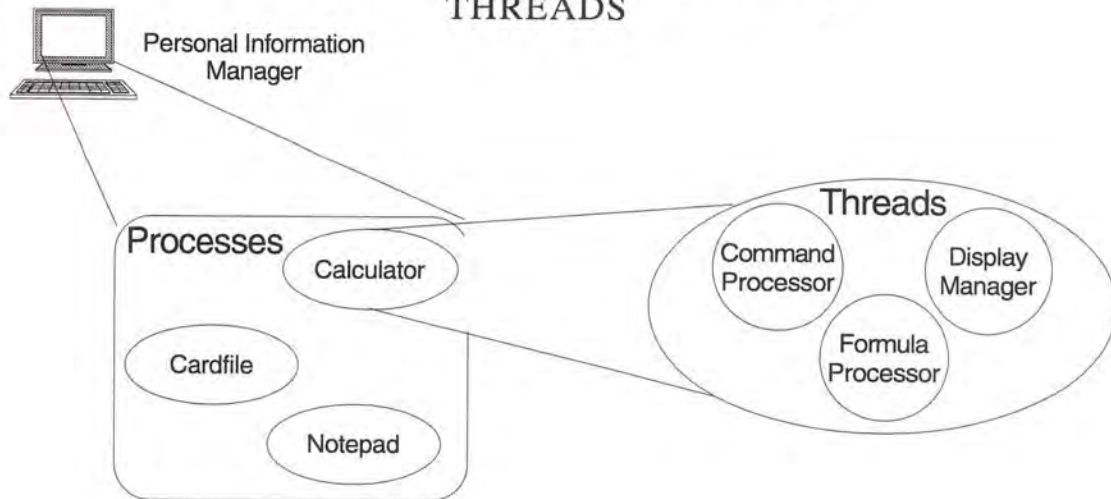
PROCESSES



- ✓ A process is the unit of ownership in OS/2.
- ✓ Memory, open files and semaphores are system resources that are owned by a process.

OS/2 Elements of Multitasking

THREADS



- ✓ A thread is a unit of work that is dispatched by the scheduler.
- ✓ Threads share the system resources owned by the process.

Thread Management

Thread Hierarchy

Creating Threads

Thread States

Thread Priority

Configuration Parameters

Global, Static and Local 'C' Variables

20THRD01 920728

— All exception handlings are done in Thread #1

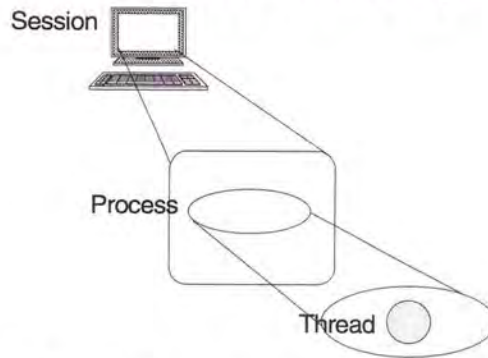
3 types:

① CTL-C

② CTL-BRK

③ KILL PROCESS

Thread Hierarchy



- ✓ A thread is a dispatchable unit of work.
- ✓ Each process has at least one thread.
- ✓ A thread owns:
 - stack segment
 - registers
 - priority
 - CPUstate
- ✓ Threads in the same process share page directories and tables.
- ✓ Threads in the same process are not protected from one another.

20THIRD02 920728

- Main procedure is thread #1
- use semaphores between threads to terminate
not DosKillThread (no cleanup)

An Example of Three Threads

```
main ()
{
    createthread ( threadA );
    createthread ( threadB );
    suspendthread (main);
}

threadA ()
{
    while ( 1 )           // do forever
    {
        putchar ( "A" );
    }
}

threadB ()
{
    while ( 1 )           // do forever
    {
        putchar ( "B" );
    }
}
```

- ① 'main' does not call threadA or threadB directly.
- ② Although there are three threads, there is only one process.

20THRD03 920728

Another Example of Three Threads

```
main ()
{
    createthread ( threadA , 'A' );
    createthread ( threadA , 'B' );
    suspendthread (main);
}

threadA (char c)
{
    while ( 1 )           // do forever
    {
        putchar ( c );
    }
}
```

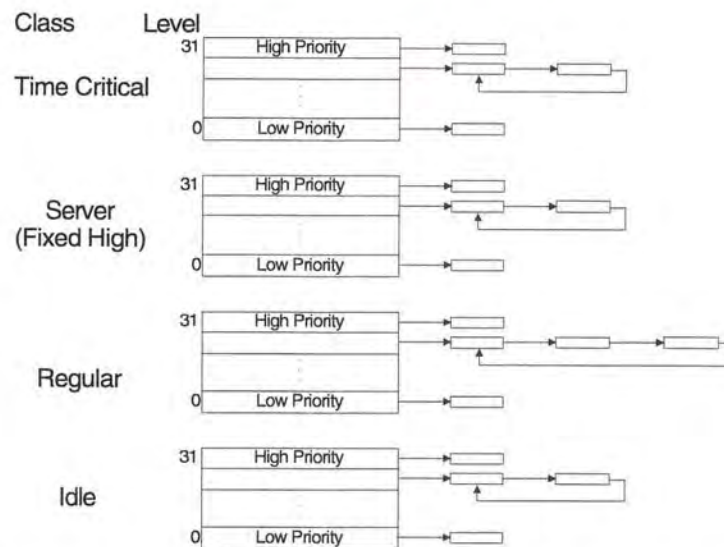
- ✓ This example shows how 2 threads can execute the same code concurrently.
- ✓ Think of 'A' and 'B' as command line parameters.
- ✓ Each thread uses the same code, but different data.

Thread States

- ✓ Once created, a thread is in one of these states:
 - Executing
 - Ready
 - Blocked
- ✓ Newly created threads may be put in the ready state or blocked state depending on the creation parameters.
- ✓ Only one thread is running at a time.
- ✓ The scheduler decides which thread will run based on the priorities of all threads ready to execute and current system load.

20THRD05 920728

Priority Classes and Levels



- ✓ The initial priority of a thread is inherited from the creating thread.
- ✓ Each thread in the system is assigned a priority class and a priority level.
- ✓ OS/2 schedules threads at the same priority in round-robin order.

20THRD06 920728

→ For idle class processes, before terminating, set priority to Regular class to get enough cpu cycles to actually exit process.

Dynamic Priority Adjustment

- ④ Dynamic priority adjustment is the scheduler's ability to alter the priority of threads in the regular class.
- ④ All threads of the foreground process receive a priority boost. The thread performing the user I/O receives an additional boost.
- ④ When a thread becomes ready to run as the result of an I/O operation completing, the priority is boosted to the highest level in the class.
- ④ A thread that is ready to run but, has not executed for the MAXWAIT period, is boosted out of its current class to a level just below time critical.
- ④ When a thread receives an I/O or starvation boost, its new priority and timeslice are retained until the thread executes for a single timeslice.

20THRID07 920728

Configuration Parameters Affecting Multitasking

(IN CONFIG.SYS)

- ✓ THREADS = 64 - 4095
Controls the number of threads that may be created
- ✓ TIMESLICE = ≥ 32 [≤ 65536]
Specifies the CPU time allocated to each thread
- ✓ PRIORITY = [ABSOLUTE | DYNAMIC]
Selects the priority calculation in scheduling Regular class threads.
- ✓ MAXWAIT = 1 - 255
Specifies the max time a thread must wait for the CPU before having its priority boosted.
- ✓ PRIORITY_DISK_IO = Yes|No
Foreground threads receive disk I/O priority.

20THFD08 920728

- disables automatic priority adjustments.

Thread Management Functions

DosCreateThread	}	Thread creation/termination
DosExit		
DosKillThread		

DosSuspendThread	}	Thread suspension/resumption
DosResumeThread		
DosEnterCritSec		
DosExitCritSec		

DosGetInfoBlocks

DosWaitThread

DosSetPriority

DosGetInfoBlocks

DosGetInfoBlocks (PTIB *ptib, --> ptr to ThreadInfoBlock1
PPIB *ppib); --> ptr to ProcessInfoBlock

struct pib_s

```
{  
    ULONG pib_ulpid;  
    ULONG pib_ulppid;  
    ULONG pib_hmte;  
    PCHAR pib_pchcmd;  
    PCHAR pib_pchenv;  
    ULONG pib_flstatus;  
    ULONG pib_ultype;  
};
```

struct tib_s

```
{  
    PVOID tib_pexchain;  
    PVOID tib_pstack;  
    PVOID tib_pstacklimit;  
    PTIB2 tib_ptib2;  
    ULONG tib_version;  
    PVOID tib_ordinal;  
};
```

struct tib2_s

```
{  
    ULONG tib2_ultid;  
    ULONG tib2_ulpri;  
    ULONG tib2_version;  
    USHORT tib2_usMCCount;  
    USHORT tib2_fMCForceFlag;  
};
```

```
PTIB ptib;  
PPIB ppib;  
TID tid;
```

```
DosGetInfoBlocks ( &ptib, &ppib )  
tid = ptib->tib_ptib2->tib2_ultid;
```

20THRD10 930419

→ Retrieves thread id.

DosCreateThread

DosCreateThread (PTID ptid, --> tid of created thread
 PFNTHREAD pfn, <-- function name
 ULONG param, <-- argument value or reference
 ULONG flag, <-- execution flag
 ULONG cbstack); <-- stack maximum size

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
115	ERROR_PROTECTION_VIOLATION
164	ERROR_MAX_THREADS_REACHED

0x0	CREATE_READY
0x1	CREATE_SUSPENDED
0x0	STACK_SPARSE
0x2	STACK_COMMITTED

- ✓ Asynchronous
- ✓ The specified stack size should include an additional page for the guard page.
- ✓ Initially, only the first two stack pages are committed with the second page a guard page unless bit 1 is set in the execution flag.
- ✓ The function being invoked must be defined with the IBM C Set/2 'system' linkage.

```
void _System readfile ( PVOID pRead );
```

```
DosCreateThread (&tid, (PFNTHREAD) readfile, (ULONG) pRead,  

  CREATE_READY | STACK_SPARSE, 8192 );
```

20THRD11 940125

- *_System* tells thread to expect params passed on the Stack.

DosEnterCritSec, DosExitCritSec

DosEnterCritSec (VOID); no other thread can run

0	NO_ERROR
484	ERROR_CRITSEC_OVERFLOW

DosExitCritSec (VOID); other threads may run

0	NO_ERROR
484	ERROR_CRITSEC_OVERFLOW
485	ERROR_CRITSEC_UNDERFLOW


20THRD12 920728

Ring #3

DosSuspendThread, DosResumeThread

DosSuspendThread (TID tid); <-- threadid

DosResumeThread (TID tid); <-- threadid



0	NO_ERROR
309	ERROR_INVALID_THREADID

- ✓ Threads suspended by DosSuspendThread must be specifically restarted with DosResumeThread.
- ✓ If multiple suspends are issued to the same thread, an equal number of resumes must be executed before the thread can be dispatched.

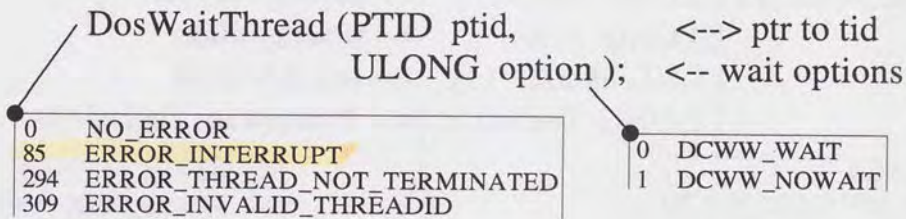
DosKillThread

DosKillThread (TID tid); <-- id of thread to terminate

0	NO_ERROR
170	ERROR_BUSY
309	ERROR_INVALID_THREADID

- ① May be used without restriction if the application was compiled with the subsystem library (/Rn).
- ① It will work with some restrictions with the multitasking libraries (/Gm) but, IS NOT RECOMMENDED.

DosWaitThread



- ✓ DosWaitThread will block until a SPECIFIC thread or ANY thread in the current process terminates.
- ✓ If TID=0, wait for any thread to terminate and return that thread's ID, otherwise, wait for specific thread.

```
APIRET rc = 0;
TID tid;

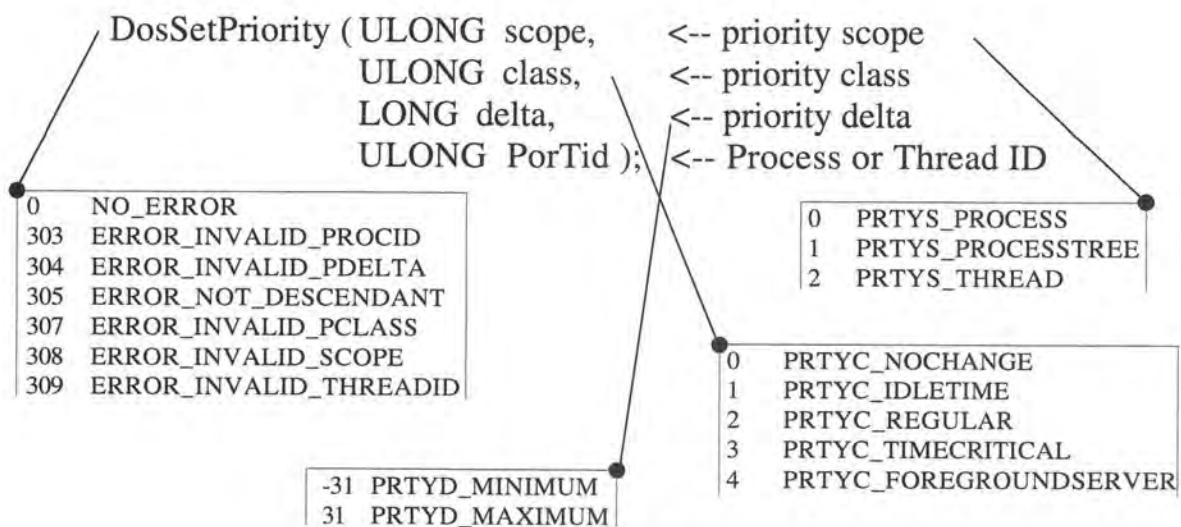
While ( !rc )
{
    tid = 0;
    rc = DosWaitThread ( &tid, DCWW_WAIT );
}
```

20THRD15 940125

- During a Long Call an 85 (ERROR interrupt) will always be issued.

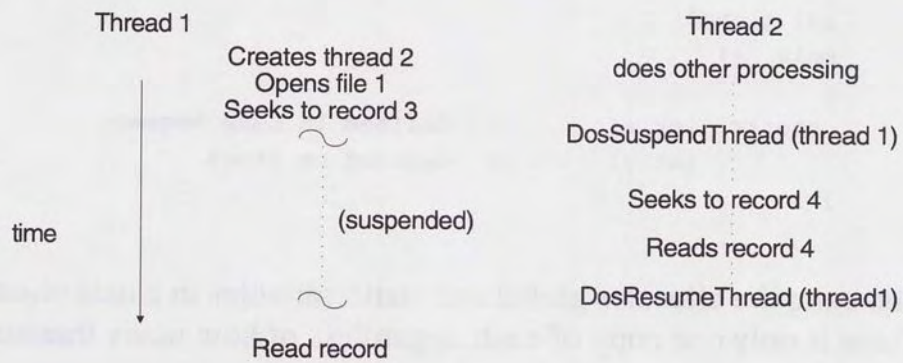
- If nowait is used & the thread you want to wait for is already running, you get a '294'

DosSetPriority



```
DosSetPriority ( PRTYS_THREAD, PRTYC_NOCHANGE, +5, tid );
```


Mutual Exclusion Problems



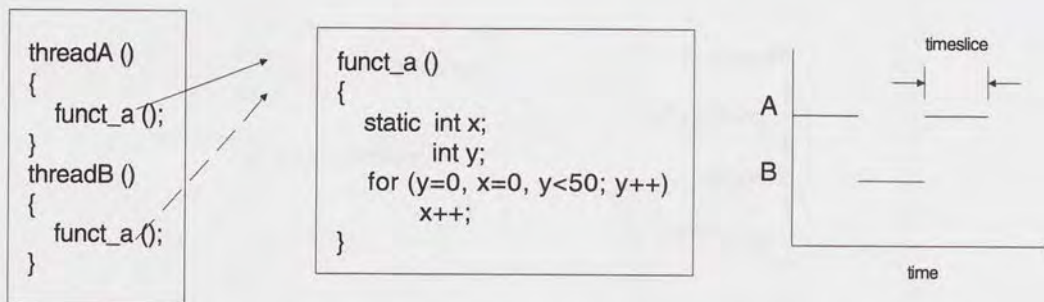
☑ Which record does thread 1 read?

Variables in C Programs

```
int global;  
main ()  
{  
    static int x;      // defined in data segment  
    int y;             // defined on stack  
}
```

- ✓ The compiler allocates global and static variables in a data object. There is only one copy of each, regardless of how many threads are defined.
- ✓ Variables defined within braces are local variables. The compiler defines these on the stack.
- ✓ Each thread has its own stack. Therefore, each thread will have a separate copy of the local variables.

Reentrancy with Multiple Threads

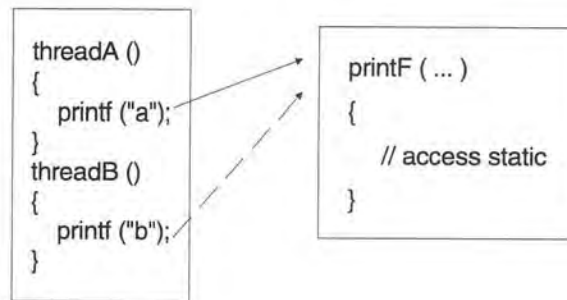


- ✓ ThreadA calls `funct_a` and then is preempted by threadB. Assume $x=y=25$.
- ✓ ThreadB calls `funct_a`. A separate copy of 'y' is allocated but, the single copy of 'x' is initially cleared. ThreadA preempts ThreadB. Assume $x=y=30$.
- ✓ ThreadA uses 'x' which is now 30, NOT 25. `funct_a` fails!
- ✓ Serializing access to the static variable through the use of a semaphore solves the problem.

20THRD19 920728

/Gm option on C compiler for re-entrant

Reentrancy with C Library Functions



- ✓ Several functions in the standard C library are not reentrant.
- ✓ Access to these functions must be serialized.
- ✓ The application must perform the serialization if multiple threads are created using the DosCreateThread function.
- ✓ Serialization will be provided by the C run-time library if threads are created using the '_beginthread' library function.

_beginthread/_endthread C Library Functions

```
int _beginthread (void (* _Optlink __thread) (void *), <-- function address
                (void *), <-- ignored by C Set/2
                unsigned, <-- stack size in bytes
                void *); <-- ptr. to argument(s)
```

_endthread (void)

- ✓ _beginthread calls DosCreateThread and then performs thread specific initialization in the context of the new thread.
- ✓ Returns tid if successful or -1 on error.
- ✓ The function invoked must use the default Optlink linkage.
- ✓ Available in the multithread library (/Gm).
- ✓ Function prototypes are contained in process.h.

```
tid = _beginthread ( readfile, NULL, 8192, (PVOID) &Arg );
if ( tid == (TID) -1 ) terminate;
```

20THRD21 930628

use this for re-entrant code
_beginthread is used most often !!!

DosCreateThread vs _beginthread

✓ DosCreateThread + Subsystem Lib (/Rn) = 😊

✓ DosCreateThread + Multi-Thread Lib (/Gm) = 😞

✓ _beginthread + Multi-Thread Lib (/Gm) = 😊

↓
DosCreateThread → Thread Specific Initialization

Synchronization Using Semaphores

Major Uses of Semaphores

Classes and Types of Semaphores

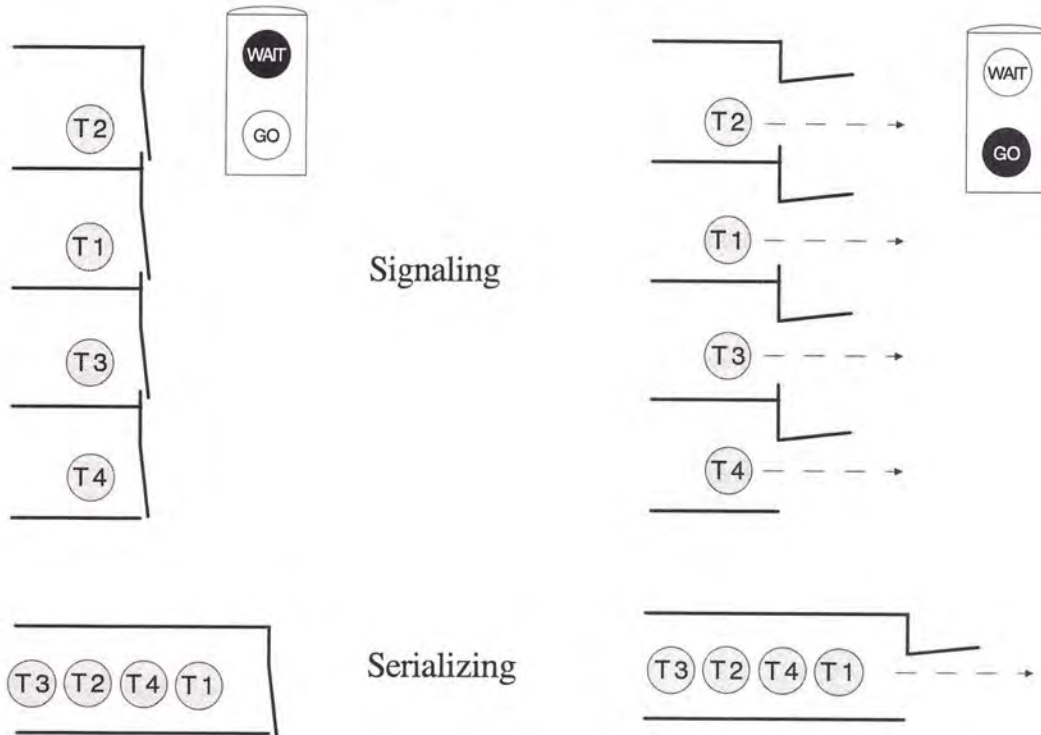
Signaling Events

Serializing the Access to Resources

Monitoring Multiple Semaphores

20SEMA01 920728

Uses of Semaphores



20SEMA02 920728

- 64,000 shared Semaphores possible

Classes and Type of Semaphores

✓ Classes:

Private - accessible by threads in the current process

Shared - accessible by threads in any process

✓ Types:

Event - used for signaling

Mutex - used for serializing

MuxWait - used for monitoring multiple Event or
Mutex semaphores

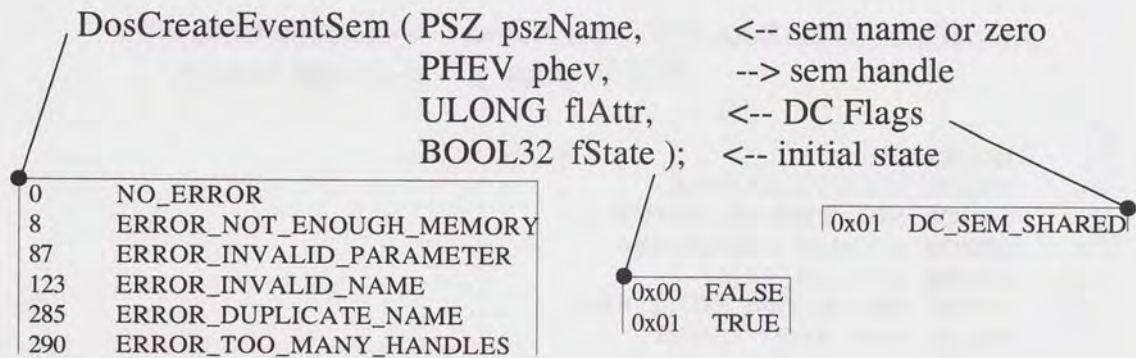
20SEMA03 920728

Application of OS/2 Semaphores

<u>Use</u>	<u>Scope</u>	<u>Semaphore Type</u>
Serialization	Between threads in different processes	Named Mutex Unnamed shared Mutex
Signaling	Between threads in different processes	Named Event Unnamed shared Event
Serialization	Between threads in one process	Unnamed Mutex
Signaling		Unnamed Event

20SEMA04 920728

DosCreateEventSem



- ✓ Named event semaphores (\SEM32\) may be shared between processes by default. Unnamed semaphores may be designated as shared.
- ✓ If the state of the semaphore is false (reset), `DosWaitEventSem` will block.
- ✓ If the state of the semaphore is true (posted), the wait function will not block.

```

#define POSTED TRUE
#define RESET  FALSE

```

```

DosCreateEventSem ( "\\SEM32\\MySem", &hev, 0, RESET );

```

20SEMA05 930206

DosOpenEventSem

DosOpenEventSem (PSZ pszName, <-- sem name or zero
PHEV phev); <--> sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME
187	ERROR_ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

- ✓ Makes available for use by ^{ANOTHER} ~~the calling~~ process an event semaphore that was previously created with the 'share' attribute.
- ✓ If opening a named event semaphore, the sem handle must be initialized to zero prior to the open.
- ✓ If opening an unnamed event semaphore, the sem handle is the value returned from the create function.

```
HEV hev = 0;
```

```
DosOpenEventSem ( "\\SEM32\\MySem", &hev );
```

20SEMA06 930125

DosCloseEventSem

DosCloseEventSem (HEV hev); <-- sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

- ✓ Decrements the open count for the process. If the open count goes to zero (0), the reference count is decremented.
- ✓ If the resulting reference count is zero (0), the semaphore is deleted from the system.

20SEMA07 930125

DosPostEventSem, DosResetEventSem

DosPostEventSem (HEV hev); <-- sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
298	ERROR_TOO_MANY_POSTS
299	ERROR_ALREADY_POSTED

- ✓ Asserts the specified event semaphore.
- ✓ Threads blocked on DosWaitEventSem will unblock.

DosResetEventSem (HEV hev, <-- sem handle
PULONG pulPostCt); --> number of posts

0	NO_ERROR
6	ERROR_INVALID_HANDLE
300	ERROR_ALREADY_RESET

- ✓ Negates the specified event semaphore, returns the post count, and resets the post count to zero.
- ✓ Threads issuing a DosWaitEventSem will block.

DosResetEventSem (hev, &ulPostCt);

20SEMA08 921020

DosWaitEventSem

```
DosWaitEventSem (HEV hev,          <-- event semaphore handle
                ULONG ulTimeout ); <-- milliseconds
```

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
640	ERROR_TIMEOUT

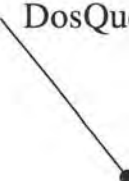
SEM_INDEFINITE_WAIT
SEM_IMMEDIATE_RETURN
or time in milliseconds

- ✓ If the event semaphore is in the reset (false) state, this function will block for the specified amount of time.
- ✓ If the event semaphore is in the posted (true) state, this function returns immediately.

```
DosWaitEventSem ( hev, 2000 );
```


DosQueryEventSem

DosQueryEventSem (HEV hev, <-- event semaphore handle
PULONG pulPostCt); --> ptr to returned post count

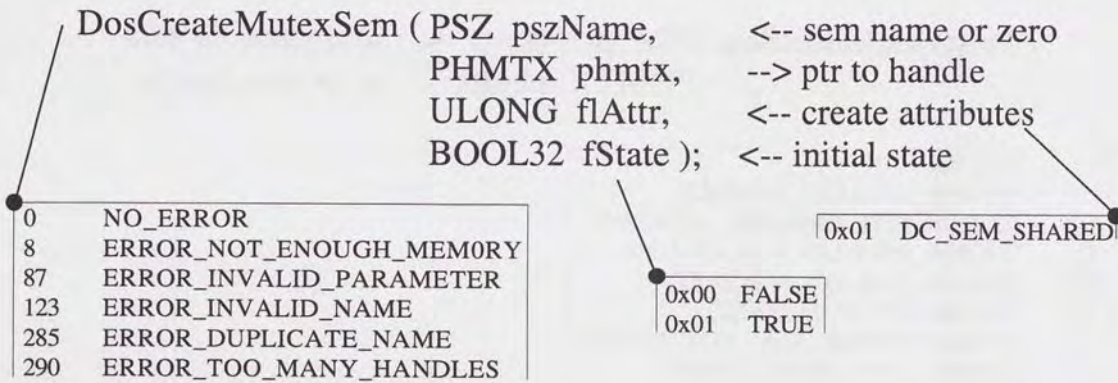


0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER

- ✓ Returns the number of times the semaphore has been posted since the last time the semaphore was in the reset state.

```
DosQueryEventSem ( hev, &puPostCt );
```

DosCreateMutexSem



- ✓ Named mutex semaphores (\SEM32\) may be shared between processes by default. Unnamed semaphores may be designated as shared.
- ✓ If created in the false state, the semaphore is not owned.
- ✓ If created in the true state, the semaphore is owned by the creating thread.

DosOpenMutexSem


DosOpenMutexSem (PSZ pszName, <-- sem name or zero
PHMTX phmtx); <--> sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
123	ERROR_INVALID_NAME
187	ERROR_ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

- ✓ Makes available for use by the calling process a mutex semaphore that was previously created with the 'share' attribute.
- ✓ If opening a named event semaphore, the sem handle should be initialized to zero prior to the open.
- ✓ If opening an unnamed event semaphore, the sem handle is the value returned from the create function.

DosCloseMutexSem

DosCloseMutexSem (HMTX hmtx); <-- sem handle



0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

- ✓ Invalidates the semaphore handle for the current process and decrements the reference count.
- ✓ If the resulting reference count is zero (0), the semaphore is deleted from the system.

20SEMA13 920728

DosRequestMutexSem, DosReleaseMutexSem

```
DosRequestMutexSem ( HMTX hmtx,          <-- semaphore handle
                    ULONG ulTimeout ); <-- milliseconds
```

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
640	ERROR_TIMEOUT

SEM_INDEFINITE_WAIT
SEM_IMMEDIATE_RETURN
or time in milliseconds

```
DosReleaseMutexSem ( HMTX hmtx ); <-- semaphore handle
```

0	NO_ERROR
6	ERROR_INVALID_HANDLE
288	ERROR_NOT_OWNER

- ④ If multiple requests are issued from the same thread, a corresponding number of releases must be issued to relinquish ownership of the semaphore

DosQueryMutexSem

DosQueryMutexSem (HMTX hmtx, <-- mutex semaphore handle
 PPID ppid, --> ptr to returned PID of owner
 PTID ptid, --> ptr to returned TID of owner
 PULONG pulCount); --> ptr to returned request count

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED

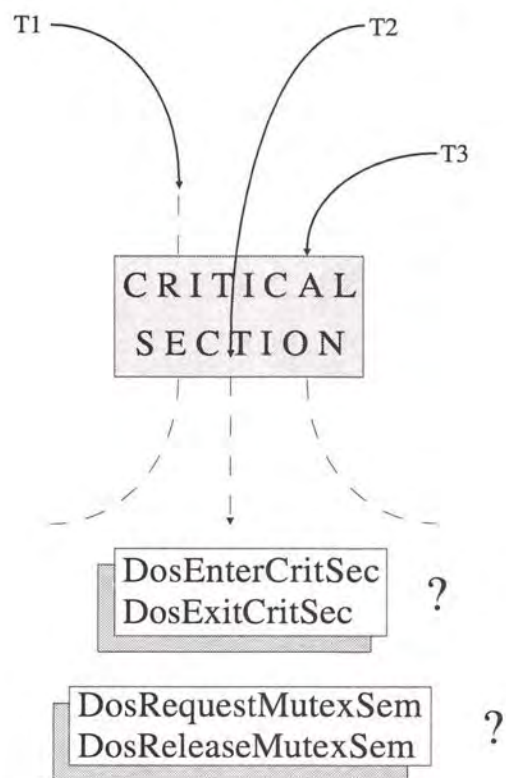
- ✓ The returned PID and TID identify the current owner of the mutex semaphore or of the previous owner that terminated without releasing the semaphore.
- ✓ The request count is the number of requests minus the number of releases.

```
PID pid;  
TID tid;  
ULONG ulCount;
```

```
DosQueryMutexSem ( hmtx, &pid, &tid, &ulCount );
```

20SEMA15 921027

CritSec vs MutexSem



20SEMA16 920722

DosCreateMuxWaitSem

DosCreateMuxWaitSem (PSZ pszName,
 PHMUX phmux,
 ULONG cSemRec,
 PSEMRECORD pSemRec,
 ULONG flAttr);

<-- sem name or zero
 --> sem handle
 <-- entry count
 <-- asr structures
 <-- DC flags

0 NO_ERROR
 6 ERROR_INVALID_HANDLE
 8 ERROR_NOT_ENOUGH_MEMORY
 87 ERROR_INVALID_PARAMETER
 100 ERROR_TOO_MANY_SEMAPHORES
 105 ERROR_SEM_OWNER_DIED
 123 ERROR_INVALID_NAME
 284 ERROR_DUPLICATE_HANDLE
 285 ERROR_DUPLICATE_NAME
 290 ERROR_TOO_MANY_HANDLES
 292 ERROR_WRONG_TYPE

0x01 DC_SEM_SHARED
 0x02 DCMW_WAIT_ANY
 0x04 DCMW_WAIT_ALL

```

typedef struct _PSEMRECORD
{
    HSEM hsemCur;
    ULONG ulUser;
} SEMRECORD, *PSEMRECORD;
    
```

- ✓ If both event and mutex semaphores are to be monitored, a separate muxwait semaphore must be created for each type.
- ✓ If the entry count is zero (0), the pSemRec must also be zero (0).

```
SEMRECORD SemRecs[16];
```

```
DosCreateMuxWaitSem ( 0, &hmux, 16, SemRecs, DC_SEM_SHARED | DCMW_WAIT_ANY );
```

20SEMA17 931005

DosOpenMuxWaitSem

DosOpenMuxWaitSem (PSZ pszName, <-- sem name or zero
PHMUX phmux); <--> sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
123	ERROR_INVALID_NAME
187	ERROR_ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

- ✓ Makes available for use by the calling process a muxwait semaphore that was previously created with the 'share' attribute.
- ✓ If opening a named muxwait semaphore, the sem handle should be zero.
- ✓ If opening an unnamed event semaphore, the sem handle is the value returned from the create function.

20SEMA18 930721

DosCloseMuxWaitSem


DosCloseMuxWaitSem (HMUX hmutex); <-- sem handle

0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

- ✓ Ends access to a muxwait semaphore for all of the threads in the calling process.
- ✓ When all processes that opened the semaphore have either closed it or have ended, the semaphore is deleted from the system.


DosAddMuxWaitSem, DosDeleteMuxWaitSem

DosAddMuxWaitSem (HMUX hmutex, <-- handle of muxwait semaphore
PSEMRECORD pSemRec);<-- ptr to SemRec to add



0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
100	ERROR_TOO_MANY_SEMAPHORES
105	ERROR_SEM_OWNER_DIED
284	ERROR_DUPLICATE_HANDLE
292	ERROR_WRONG_TYPE

DosDeleteMuxWaitSem (HMUX hmutex, <-- mux sem handle
HSEM hsem); <-- handle of semaphore to be deleted



0	NO_ERROR
6	ERROR_INVALID_HANDLE
286	ERROR_EMPTY_MUXWAIT

DosWaitMuxWaitSem

```
DosWaitMuxWaitSem (HMUX hmutex,          <-- mux sem handle
                   ULONG ulTimeout,        <-- milliseconds
                   PULONG pulUser );      --> from PSEMRECORD
```

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
286	ERROR_EMPTY_MUXWAIT
287	ERROR_MUTEX_OWNED
292	ERROR_WRONG_TYPE
640	ERROR_TIMEOUT

-1L	SEM_INDEFINITE_WAIT
0L	SEM_IMMEDIATE_RETURN

or time in milliseconds

- ① If the muxwait semaphore was created with the WAIT_ANY option, the SemRecord is from the semaphore that was posted. If mutex semaphores are in the list, the released semaphore will be owned by the caller.
- ① If the WAIT_ALL option was used, the SemRecord is from the last semaphore posted. If mutex semaphores are in the list, all will be owned by the caller.

20SEMA21 920722

DosQueryMuxWaitSem

```

DosQueryMuxWaitSem (
    HMUX hmutex,          <-- mux sem handle to query
    PULONG pcSemRec,      <--> ptr to max list size
    PSEMRECORD pSemRec,   --> ptr to list of semaphores
    PULONG pflAttr );    -- > DC flags from DosCreateMuxWaitSem
    
```

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
289	ERROR_PARAM_TOO_SMALL

0x01	DC_SEM_SHARED
0x02	DCMW_WAIT_ANY
0x04	DCMW_WAIT_ALL

- ✓ Retrieves the semaphore records from a muxwait semaphore list.
- ✓ pSemRec points to a buffer that will receive the semaphore records.
- ✓ On input, pcSemRec points to the max number of records pSemRec can hold. On return, it points to the number of records retrieved.

```

ULONG cSemRec = 64;
SEMRECORD semrec[cSemRec];
ULONG flAttr;
    
```

```

DosQueryMuxWaitSem ( hmutex, &cSemRec, semrec, &flAttr );
    
```

20SEMA22 931095

Summary of Semaphore API's

DosCreateMutexSem
DosOpenMutexSem
DosCloseMutexSem
DosRequestMutexSem
DosReleaseMutexSem
DosQueryMutexSem
DosCreateEventSem
DosOpenEventSem
DosCloseEventSem
DosResetEventSem

DosPostEventSem
DosWaitEventSem
DosQueryEventSem
DosCreateMuxWaitSem
DosOpenMuxWaitSem
DosCloseMuxWaitSem
DosWaitMuxWaitSem
DosAddMuxWaitSem
DosDeleteMuxWaitSem
DosQueryMuxWaitSem

Notes

Timers

Synchronous Timers

Asynchronous Timers

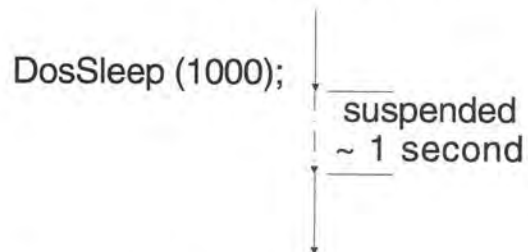
20TIMR01 921008

DosSleep

DosSleep (ULONG msec); <-- interval in milliseconds

0	NO_ERROR
322	ERROR_TS_WAKEUP

Thread Execution



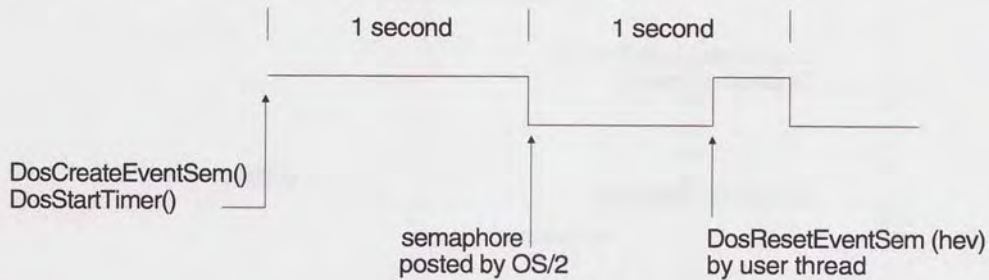
- ✓ Suspends the current thread for a specified time interval.
- ✓ If the time interval is zero (0), the thread relinquishes the remainder of its time slice to another ready thread of equal or higher priority. If there is no other ready thread of equal or higher priority, the call returns immediately.

20TIMR02 021021

DosStartTimer

DosStartTimer (ULONG msec, <-- interval in milliseconds
 HEV hev, <-- event semaphore handle
 PHTIMER phtimer); --> timer handle

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER



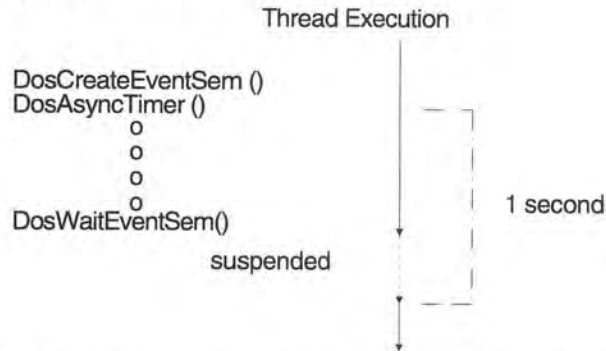
- ✓ Starts an asynchronous, repeated-interval timer and post the event semaphore each time the interval expires.
- ✓ The returned timer handle is used with DosStopTimer.

20TIMR03 921027

DosAsyncTimer

DosAsyncTimer (ULONG msec, <-- interval in milliseconds
 HEV hev, <-- event semaphore handle
 PHTIMER phtimer); --> timer handle

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER




- ✓ Starts an asynchronous, single-interval timer and post the event semaphore when the interval expires.
- ✓ The returned timer handle is used with DosStopTimer.

20TIMR04 921021

DosStopTimer

DosStopTimer (HTIMER htimer); <-- timer handle



0	NO_ERROR
326	ERROR_TS_HANDLE

- ✓ Stops either a repeated-interval or single-interval timer.

20TIMR05 920723

Notes

Process Management

Creating Processes

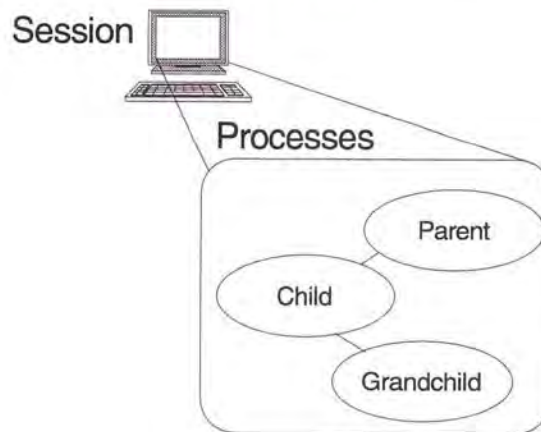
Controlling Processes

Checking Process Status

Performing Cleanup When a Process Ends

20PRCS01 921008

Processes



- ✓ A process is created by the invocation of a particular program through the DosExecPgm system call.
- ✓ Each process has its own memory, threads, file system and inter-process communication data structures which isolate it from other processes.
- ✓ Each process is initially created with a single thread. Additional threads are created through the DosCreateThread system call.

20PRCS02 920729

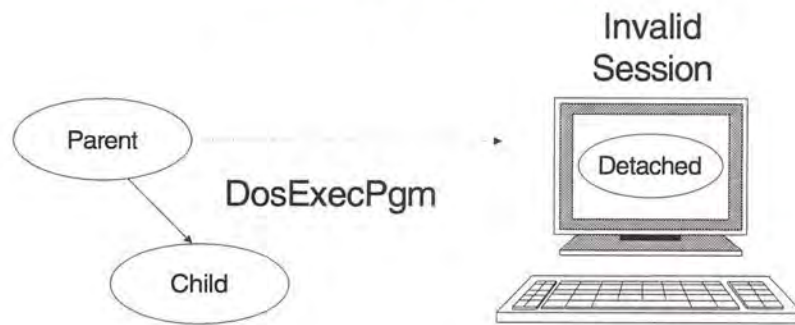
Process Resources

Environment	Semaphores
Commandline	Threads
LDT	Child processes
All allocated segments	Process ID
File handles	Maximum file handle count
Pipes	Default drive, directories
Queues	

- ✓ A process is a collection of one or more threads and associated system resources.
- ✓ A unit of ownership.
- ✓ Can be created at the commandline by:

<program name>	runs in the current session
DETACH <program name>	runs in the Invalid session
START <program name>	runs in a new session

Creating A Process



- ✓ Processes are created from the command line or under program control with the `DosExecPgm`.
- ✓ The parent process receives the child process ID number (PID) from the `DosExecPgm` function.
- ✓ Detached (daemon) processes execute in the Invalid session and may not use standard functions to communicate with the user.

DosExecPgm

DosExecPgm (PCHAR pObjname, --> ptr to message buffer
 LONG cbObjname, <-- length of message buffer
 ULONG execFlag, <-- execution flags
 PSZ pArg, <-- ptr to argument strings
 PSZ pEnv, <-- ptr to environment strings
 PRESULTCODES pRes, --> ptr to result codes buffer
 PSZ pName); <-- ptr to program file name

```

0 NO_ERROR
1 ERROR_INVALID_FUNCTION
2 ERROR_FILE_NOT_FOUND
3 ERROR_PATH_NOT_FOUND
4 ERROR_TOO_MANY_OPEN_FILES
5 ERROR_ACCESS_DENIED
8 ERROR_NOT_ENOUGH_MEMORY
10 ERROR_BAD_ENVIRONMENT
11 ERROR_BAD_FORMAT
13 ERROR_INVALID_DATA
26 ERROR_NOT_DOS_DISK
32 ERROR_SHARING_VIOLATION
33 ERROR_LOCK_VIOLATION
36 ERROR_SHARING_BUFFER_EXCEEDED
89 ERROR_NO_PROC_SLOTS
95 ERROR_INTERRUPT
108 ERROR_DRIVE_LOCKED
127 ERROR_PROC_NOT_FOUND
182 ERROR_INVALID_ORDINAL
190 ERROR_INVALID_MODULETYPE
191 ERROR_INVALID_EXE_SIGNATURE
192 ERROR_EXE_MARKED_INVALID
195 ERROR_INVALID_MINALLOCSIZE
196 ERROR_DYNLINK_FROM_INVALID_RING
  
```

RETURNED IF EXEC_SYNC ON CHILD TERM.

```

typedef struct _RESULTCODES{
  ULONG codeTerminate;
  ULONG codeResult;
} RESULTCODES,
*PRESULTCODES;
  
```

```

0 EXEC_SYNC ← WAITS TILL CHILD IS DONE
1 EXEC_ASYNC ← PID OF CHILD IS RETURNED
2 EXEC_ASYNCRESULT ← MUST USE THIS IF YOU WANT TO USE: DosWaitChild
3 EXEC_TRACE
4 EXEC_BACKGROUND
5 EXEC_LOAD
6 EXEC_ASYNCRESULTDB
  
```

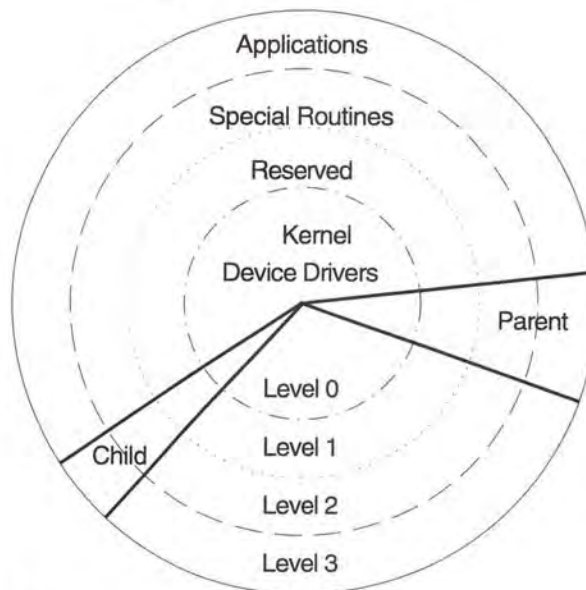
```

DosExecPgm ( Objbuf, 254, EXEC_ASYNCRESULT, 0,
0, &Resc, "readfile.exe" );
  
```

- ✓ If the process is executed synchronously, codeTerminate and codeResult are valid upon return from the function.
- ✓ If the executed asynchronously, codeTerminate contains the process ID.

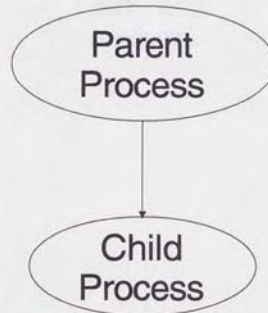
20PFC505 940119

Process Isolation



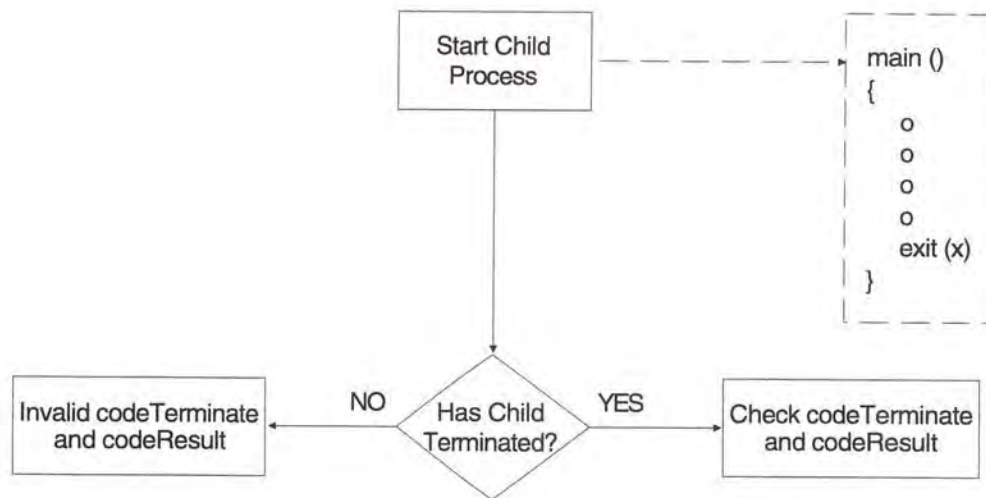
- ✓ Parent and child processes are isolated by separate page tables and directories.
- ✓ Files and pipes are inherited by the child process.
- ✓ Parent process can control STDIN, STDOUT, and STDERR file handles.

Controlling A Child Process



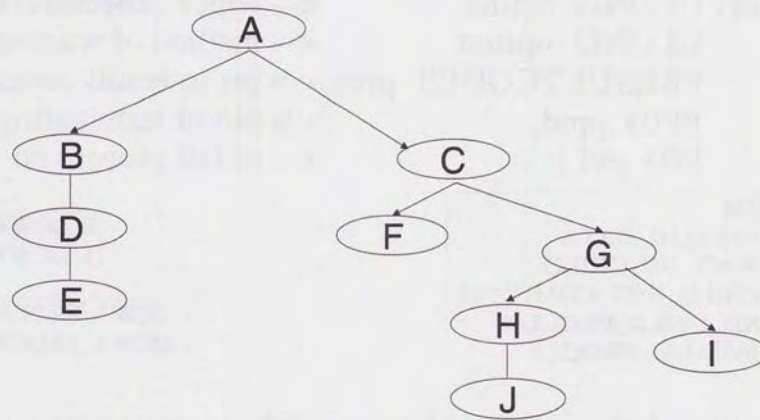
- ✓ After starting a child process, the parent can:
 - terminate the child
 - change the priority of all threads in the child
 - test or wait for child completion

Process Concurrency



- ✓ Checking for termination may occur at any time prior to or following the 'exit ()'. codeTerminate and codeResult are only valid after the 'exit ()'.

Command Subtrees



- ✓ DosWaitChild, DosKillProcess and DosSetPriority have two forms:
- ProcessID - only the direct child is affected
 - Subtree - the direct child and all descendents are affected

Testing for Child Process Termination

`DosWaitChild (ULONG action,`
 `ULONG option,`
 `PRESULTCODES pres,--> ptr to result codes`
 `PPID ppid,`
 `PID pid);`

<-- which process(s) on which to wait
 <-- method of waiting
 --> pid of terminating process
 <-- pid of process on which to wait

0	NO_ERROR
13	ERROR_INVALID_DATA
128	ERROR_WAIT_NO_CHILD
129	ERROR_CHILD_NOT_COMPLETE
184	ERROR_NO_CHILD_PROCESS
303	ERROR_INVALID_PROCID

0	DCWW_WAIT
1	DCWW_NOWAIT

0	DCWA_PROCESS
1	DCWA_PROCESSTREE

- ✓ Waits for completion of a child process whose execution is asynchronous to that of its parent process.
- ✓ The child process is created with the `execFlag = EXEC_ASYNCRESULT`.
- ✓ If the child process has multiple threads, the result code returned is the one passed by the `DosExit` request that ends the process.

```
DosWaitChild ( DCWA_PROCESS, DCWW_WAIT, &Resc, &pid, Resc.codeTerminate );
```

20PRCS10 940119

DosWaitChild: Process Subtrees

Action Code	Wait Option	PID	Action
DCWA_PROCESSTREE	DCWW_WAIT	n	Wait until the process subtree has completed and then return the direct child's termination code.
DCWA_PROCESSTREE	DCWW_NOWAIT	n	If the process subtree has completed, return the direct child's termination code. Otherwise, return ERROR_CHILD_NOT_COMPLETE.

Return Codes:

ERROR_WAIT_NO_CHILDREN = 128

ERROR_CHILD_NOT_COMPLETE = 129

NO_ERROR = normal completion; codeTerminate and codeResult are valid

- ✓ These will continue to work correctly even if the child process is changed to use more/less child processes of its own.

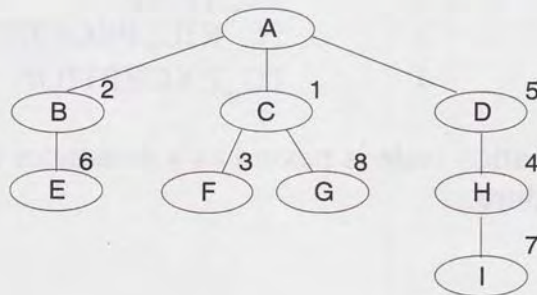
DosWaitChild: Individual Processes

Action Code	Wait Option	PID	Action
DCWA_PROCESS	DCWW_WAIT	0	Returns as soon as ANY direct child process terminates. Returns immediately if child had already terminated.
DCWA_PROCESS	DCWW_WAIT	n	Returns as soon as the direct child 'n' terminates. Returns immediately if child had already terminated.
DCWA_PROCESS	DCWW_NOWAIT	0	Checks for ANY terminated direct child process. If one is found, its status is returned, If none found, returns ERROR_CHILD_NOT_COMPLETE
DCWA_PROCESS	DCWW_NOWAIT	n	Checks the status of direct child 'n'. If terminated, returns its status. If still running, returns ERROR_CHILD_NOT_COMPLETE.

- ✓ Use these only if the child process is part of the same application as the parent. Parent must know exactly what the child is doing.

DosWaitChild: Process Subtrees

Action Code	Wait Option	PID	Action
DCWA_PROCESSTREE	DCWW_WAIT	0	Wait until ANY direct child and all of its descendants have terminated. Selects the subtree of the first direct child that terminates.
DCWA_PROCESSTREE	DCWW_NOWAIT	0	Returns error 129 if any process in any subtree is still executing. If all subtrees have terminated, returns direct child's exit code. To wait for all subtrees, repeat call until error code 128 is returned.



20PRCS13 930925

Performing Cleanup When a Process Ends

- ✓ For each process, OS/2 maintains a list of procedures that it will call when the process terminates.
- ✓ These procedures get control under the following conditions:

Termination Code	Reason
0	TC_EXIT
1	TC_HARDERROR
2	TC_TRAP
3	TC_KILLPROCESS
4	TC_EXCEPTION

- ✓ The termination code is passed as a parameter to the Exit List procedure.

Exit List Sequence

```
main ()
{
    DosExitList (0x0100 | EXLST_ADD, (PFNEXITLIST) ShutDown);
    o
    o
    o
    o
    exit (0);
}

void APIENTRY ShutDown (int TermCode)
{
    o
    o
    o
    DosExitList (EXLST_EXIT, 0);
}
```

- ✓ Exit list routines should be short and safe.
- ✓ The routines can reside in a Dynamic Link Library module.
- ✓ Ownership of mutex semaphores is transferred to the thread performing the exit procedure.

DosExitList

DosExitList (ULONG ordercode, <-- function request code
PFNEXITLIST pfn); <-- address of handler

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
8	ERROR_NOT_ENOUGH_MEMORY
13	ERROR_INVALID_DATA

Low Word, Low Byte:
1 EXLST_ADD
2 EXLST_REMOVE
3 EXLST_EXIT
Low Word, High Byte:
Invocation order
High Word = 0

- ✓ Defines a routine(s) (EXLST_ADD) to be given control when a process completes execution.
- ✓ Multiple routines with the same invocation order will be executed in a LIFO order.
- ✓ DosExitList with the 'ordercode' = EXLST_EXIT should be the last function in the exit list routine. This returns control to the kernel.

Shared Memory Objects

Named (Global) Shared Memory

Unnamed (Local) Shared Memory

20SHRM01 920723

Two Types of Shared Objects

✓ Global (named)

Use DosAllocSharedMem & DosGetNamedSharedMem.

Any process that knows the object name may access the object (no IPC required).

✓ Private (unnamed)

Use DosAllocSharedMem and DosGiveSharedMem OR DosGetSharedMem.

Interprocess communication is necessary to pass pointers and PID's.

Accessibility is limited to participating processes.

Using Named Shared Objects

Process A

```
memname = \SHAREMEMMyObj
```

```
// create the shared object and  
// increment the reference count
```

```
DosAllocSharedMem ( ppb, memname, size, allocflg);
```

```
o
```

```
o
```

```
o
```

```
// decrement the reference count
```

```
DosFreeMem ( pb );
```

Process B

```
memname = \SHAREMEMMyObj
```

```
// enable access to the shared object and  
// increment the reference count
```

```
DosGetNamedSharedMem ( ppb, memname, flag );
```

```
o
```

```
o
```

```
o
```

```
// decrement the reference count
```

```
DosFreeMem ( pb );
```

- ✓ Each `DosGetNamedSharedMem` increments the object's reference count.
- ✓ Each `DosFreeMem` decrements the reference count.
- ✓ When the reference count = 0, OS/2 discards the object.

NOTE: Process A & B must synchronize so that Process A does not free the object before Process B gets the object.

20SHRIM03 920807

Using Unnamed Shared Objects

Process A

```
// allocate the object for sharing
DosAllocSharedMem ( ppb, 0, size, allocflg );
// obtain Process B's PID...
...
; activate the object for Process B
DosGiveSharedMem ( pb, pid, flag );
; pass the pointer to Process B
; using IPC
...
; release the object
DosFreeMem ( pb );
```

Process B

```
// obtain shared object
// pointer via IPC
...
// access the object
...
// release the object
DosFreeMem ( pb );
```

- ✓ Each DosGiveSharedMem (DosGetSharedMem) increments the reference count.
- ✓ As with named objects, when the reference count = 0, OS/2 discards the object.

NOTE: Process B could use DosGetSharedMem instead. Process A must still pass the pointer to Process B.

DosAllocSharedMem

DosAllocSharedMem (PPVOID ppb, --> address pointer
 PSZ pszName, <-- name or zero
 ULONG cb, <-- size in bytes
 ULONG flag); <-- allocation flags

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
123	ERROR_INVALID_NAME
183	ERROR_ALREADY_EXISTS

PAG_COMMIT	0x00000010
OBJ_TILE	0x00000040
OBJ_GETTABLE	0x00000100
OBJ_GIVEABLE	0x00000200
fPERM	(PAG_EXECUTE+PAG_READ+PAG_WRITE)
fSHARE	(OBJ_GETTABLE+OBJ_GIVEABLE)
fALLOCSHR	(OBJ_TILE+PAG_COMMIT+fSHARE+fPERM)

```
PVOID pArray3;
PSZ pszName = "\\sharemem\\MyObj";
ULONG ulAllocFlags = PAG_READ | PAG_WRITE | PAG_COMMIT;

DosAllocSharedMem ( &pArray3, pszName, 0x20000, ulAllocFlags );
```

20SHRM05 921021

DosGetNamedSharedMem

DosGetNamedSharedMem (PPVOID ppb, --> address pointer
 PSZ pszName, <-- name
 ULONG flag); <-- access requested

0 NO_ERROR
 2 ERROR_FILE_NOT_FOUND
 8 ERROR_NOT_ENOUGH_MEMORY
 87 ERROR_INVALID_PARAMETER
 95 ERROR_INTERRUPT
 123 ERROR_INVALID_NAME
 212 ERROR_LOCKED

PAG_READ 0x00000001
 PAG_WRITE 0x00000002
 PAG_EXECUTE 0x00000004
 PAG_GUARD 0x00000008
 fPERM (PAG_EXECUTE+PAG_READ+PAG_WRITE)
 fGETNMSHR (fPERM)

- ✓ Enables access to a named memory object that was allocated by DosAllocSharedMem.
- ✓ The initial call by a process to this function increments the object's reference count.
- ✓ At least one process must call this function prior to the allocating process calling DosFreeMem, else the memory object will be released.

```
PSZ pszMemName = "\\sharemem\\MyObj";
PVOID pArray3;
```

```
DosGetNamedSharedMem ( &pArray3, pszMemName, PAG_READ );
```

20SHRM06 930628

DosGiveSharedMem

DosGiveSharedMem (PVOID pb, <-- identify by address
 PID pid, <-- PID of recipient
 ULONG flag); <-- recipient access rights

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
212	ERROR_LOCKED
303	ERROR_INVALID_PROCID
487	ERROR_INVALID_ADDRESS

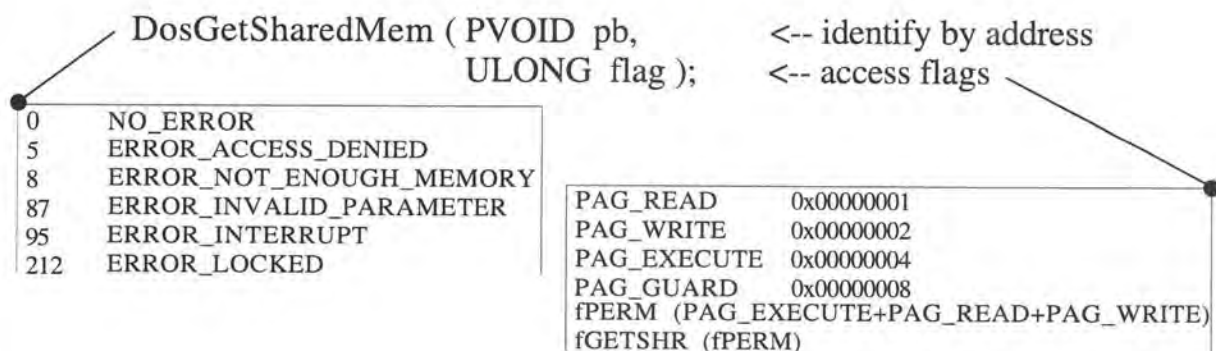
PAG_READ	0x00000001
PAG_WRITE	0x00000002
PAG_EXECUTE	0x00000004
PAG_GUARD	0x00000008
fPERM (PAG_EXECUTE+PAG_READ+PAG_WRITE)	
fGIVESH (fPERM)	

- ✓ Gives another process access to an unnamed memory object allocated 'giveable' with DosAllocSharedMem.
- ✓ The initial call by a process to this function increments the object's reference count.

```
DosGiveSharedMem ( pb, pid, PAG_READ | PAG_WRITE );
```

20SHRM07 930628

DosGetSharedMem



- ✓ Enables access to an unnamed memory object that was allocated 'gettable' with DosAllocSharedMem.
- ✓ The initial call to this function by a process increments the object's reference count.
- ✓ At least one process must call this function prior to the allocating process calling DosFreeMem, else the memory object will be released.

```
DosGetSharedMem ( pb, PAG_READ | PAG_WRITE );
```


File Input/Output

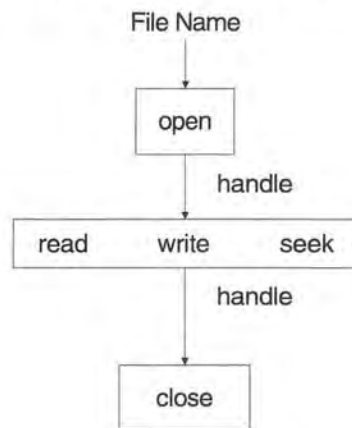
OS/2 File Systems

Opening/Creating/Accessing Files

Standard and Extended File Attributes

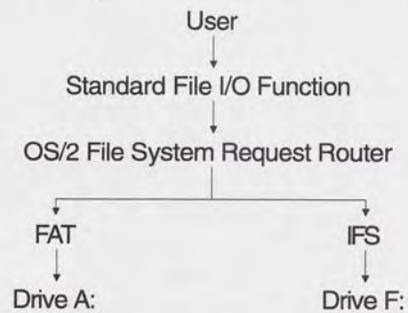
20FILE01 920807

OS/2 File Systems



- ✓ Applications use file systems to manage storage devices.
- ✓ To an application, files are logical sequences of data.
- ✓ File systems manage the physical location of data on the storage device.
- ✓ A connection must be established with the file or device to perform input/output.

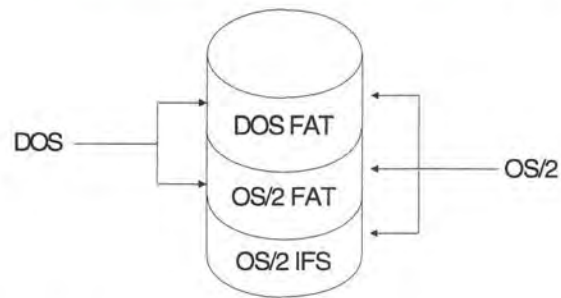
Multiple File Systems



- ✓ OS/2 supports the coexistence of multiple file systems:
 - OS/2 FAT file system similar to DOS (default)
 - Installable File Systems (IFS)
- ✓ Each logical device may be managed by a different file system.
- ✓ A File System Driver (FSD) must be loaded for each file system other than the FAT system.
- ✓ The FSD is installed via the IFS statement in CONFIG.SYS.
- ✓ The device is managed by the file system used to format the device.

20FILE03 920807

Recognizing DOS and OS/2 Files



- ✓ OS/2 recognizes files created by:
 - DOS FAT file system
 - OS/2 FAT file system
 - Installable File Systems
- ✓ DOS recognizes files created by:
 - DOS FAT file system
 - OS/2 FAT file system
- ✓ The primary partition must be FAT for dual-booting.

Naming OS/2 Files

✓ FAT File Systems:

8 character file name

3 character extension

Names are case insensitive

✓ Installable File Systems:

Each element of a full path name may be a max of 254 characters.

Names are not case sensitive, but case is preserved.

Blanks in the middle of a file name are significant.

Additional legal characters for file names + - ; , []

`This.file.name.would.have.been.longer.but,.I.got.tired.of.typing`

20FILE05 920807

Using Long File Names

- ✓ Programs that can handle long file names are marked as being 'long-name-aware'.
- ✓ This is done with the Module Definition File by including the `NEWFILES` statement.
- ✓ For compatibility, DLL's should not use or return long file names.
- ✓ Files with long names may be moved to non-IFS disks using the File Manager.
- ✓ The file name will be truncated to a max of 11 characters and the original long name will be stored in an Extended Attribute.
- ✓ When copied back to an IFS drive, the long name will be restored.

Opening/Creating Files

- ✓ A file must be opened before it can be accessed.
- ✓ The DosOpen function will open an existing file or create a new file.
- ✓ The file is opened/created by name and a file handle is returned.
- ✓ The handle is used to reference the file in other file system functions.
- ✓ When open/creating a file, the caller specifies how the file will be accessed and how it will be shared with other processes.

Access Mode

Read Only
Write Only
Read/Write

Share Mode

Deny All (no sharing)
Deny Read
Deny Write
Deny None (full sharing)

20FILE07 920807

DosOpen

DosOpen (PSZ pszFileName, <-- file name
 PHFILE pHf, --> ptr to returned file handle
 PULONG pulAction, --> ptr to returned action taken
 ULONG cbFile, <-- size in bytes for create or replace
 ULONG ulAttribute, <-- file attributes
 ULONG fsOpenFlags, <-- open flags
 ULONG fsOpenMode, <-- open mode
 PEAOP2 peaop2); <-- ptr to EA structure

0 NO_ERROR
 2 ERROR_FILE_NOT_FOUND
 3 ERROR_PATH_NOT_FOUND
 4 ERROR_TOO_MANY_OPEN_FILES
 5 ERROR_ACCESS_DENIED
 12 ERROR_INVALID_ACCESS
 26 ERROR_NOT_DOS_DISK
 32 ERROR_SHARING_VIOLATION
 36 ERROR_SHARING_BUFFER_EXCEEDED
 82 ERROR_CANNOT_MAKE
 87 ERROR_INVALID_PARAMETER
 99 ERROR_DEVICE_IN_USE
 108 ERROR_DRIVE_LOCKED
 110 ERROR_OPEN_FAILED
 112 ERROR_DISK_FULL
 206 ERROR_FILENAME_EXCED_RANGE
 231 ERROR_PIPE_BUSY

0x0000 OPEN_ACCESS_READONLY
 0x0001 OPEN_ACCESS_WRITEONLY
 0x0002 OPEN_ACCESS_READWRITE
 0x0010 OPEN_SHARE_DENYREADWRITE
 0x0020 OPEN_SHARE_DENYWRITE
 0x0030 OPEN_SHARE_DENYREAD
 0x0040 OPEN_SHARE_DENYNONE
 0x0080 OPEN_FLAGS_NOINHERIT
 0x0000 OPEN_FLAGS_NO_LOCALITY
 0x0100 OPEN_FLAGS_SEQUENTIAL
 0x0200 OPEN_FLAGS_RANDOM
 0x0300 OPEN_FLAGS_RANDOMSEQUENTIAL
 0x1000 OPEN_FLAGS_NO_CACHE
 0x2000 OPEN_FLAGS_FAIL_ON_ERROR
 0x4000 OPEN_FLAGS_WRITE_THROUGH
 0x8000 OPEN_FLAGS_DASD

0x1 FILE_EXISTED
 0x2 FILE_CREATED
 0x3 FILE_REPLACED

0x0 FILE_NORMAL
 0x1 FILE_READONLY
 0x2 FILE_HIDDEN
 0x4 FILE_SYSTEM
 0x10 FILE_DIRECTORY
 0x20 FILE_ARCHIVED

0x00 OPEN_ACTION_FAIL_IF_EXISTS
 0x01 OPEN_ACTION_OPEN_IF_EXISTS
 0x02 OPEN_ACTION_REPLACE_IF_EXISTS
 0x00 OPEN_ACTION_FAIL_IF_NEW
 0x10 OPEN_ACTION_CREATE_IF_NEW

DosOpen ("TestFile", &Hf, &ulAction, 0, FILE_NORMAL, OPEN_ACTION_OPEN_IF_EXISTS,
 OPEN_ACCESS_READONLY | OPEN_SHARE_DENYWRITE, 0);

20FILE08 921021

Synchronous File Input/Output

DosRead [Write] (HFILE hFile, <-- open file handle
PVOID pBuffer, <-- ptr to data buffer
ULONG cbRead [Write], <-- # of bytes to transfer
PULONG pcbActual); --> ptr to returned actual bytes

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
19	ERROR_WRITE_PROTECT
26	ERROR_NOT_DOS_DISK
29	ERROR_WRITE_FAULT
33	ERROR_LOCK_VIOLATION
109	ERROR_BROKEN_PIPE
234	ERROR_MORE_DATA

- ✓ DosRead (Write) automatically moves the file pointer.
- ✓ Reading and writing 512 bytes at a time (sector size) will speed up the processing
- ✓ If read returns cbActual = 0, end of file was reached.
- ✓ Standard device handle values:
0= STDIN 1 = STDOUT 2 = STDERR

20FILE09 921021

Asynchronous File Input/Output

```
typedef struct _RW
{
    HFILE hFile;
    PVOID pBuffer;
    ULONG cb;
    PULONG pcbActual;
    APIRET rc;
    HEV hev;
} RW, *PRW;

int main ( void ){
    RW rw;

    DosCreateEventSem ();
    DosCreateThread ( ptid, AsyncRW, &rw, ready, 16384 );

    AsyncRW ( PRW prw )
    {
        prw->rc = DosRead ( prw->hFile,
                           prw->pBuffer,
                           prw->cb,
                           prw->pcbActual );

        Other Processing

        DosWaitEventSem ( )
        if (rw.rc) terminate;

        Data is Valid
    }

    DosPostEventSem ();
}
```

- ✓ The event semaphore is posted when the read or write is completed.
- ✓ main must wait until the semaphore is posted to access the data in the buffer.

Changing the Size of a File

`DosSetFileSize (HFILE hFile, <-- open file handle
 ULONG cbSize); <-- size in bytes`

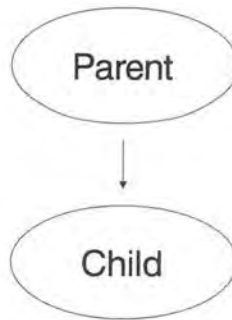
0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

- ✓ Changes the amount of space allocated to a file.
- ✓ The file must be opened for write access.
- ✓ The file can be truncated or extended in size. If extended, the value of the new bytes is undefined.

```
DosSetFileSize ( hf, 50000 );
```

20FILE11 921021

Inheriting Files



- ✓ If the parent opens a file with the inheritance option, then that file handle may be used by the child process.
- ✓ All sharing and access modes are preserved.
- ✓ Processes own files, threads do not. Any thread in a process can read from a file, even if it is not shared.

20FILE12 920808

Locking Regions of a File

```
DosSetFileLocks (HFILE hfile,          <-- open file handle
                  PFILELOCK pflUnlock, <-- ptr to unlock range
                  PFILELOCK pflLock,   <-- ptr to lock range
                  ULONG timeout,       <-- time in milliseconds
                  ULONG flags );       <-- atomic/share flags
```

```
0  NO_ERROR
6  ERROR_INVALID_HANDLE
33 ERROR_LOCK_VIOLATION
36 ERROR_SHARING_BUFFER_EXCEEDED
87 ERROR_INVALID_PARAMETER
95 ERROR_INTERRUPT
174 ERROR_ATOMIC_LOCK_NOT_SUPPORTED
175 ERROR_READ_LOCKS_NOT_SUPPORTED
```

```
_typedef struct _FILELOCK {
    LONG lOffset
    LONG lRange
} FILELOCK, *PFILELOCK
```

```
0x1 Share for read-only
0x2 Atomic lock
```

- ✓ A process may temporarily be prohibited from accessing certain areas of a file that was opened for sharing by another process.
- ✓ If processes are not accessing the same region at the same time, they both operate in parallel.
- ✓ Locked regions are not inherited from the parent process.

```
FILELOCK filelock;
filelock.lOffset = 1500;
filelock.lRange = 250;
DosSetFileLocks ( hFile, 0, &filelock, 300, 0 );
```

20FILE13 921021

Updating File Data/Directory

DosClose (HFILE hFile); <-- open file handle

DosResetBuffer (HFILE hFile); <-- open file handle

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE

- ✓ DosClose flushes the internal buffers, updates the directory and invalidates the file handle for the calling process.
- ✓ DosResetBuffer flushes internal buffers and updates the directory without closing file. If the file handle 0xFFFF is specified, all buffers for all files open to the process are flushed.

Standard File Attributes

Date and Time of Creation
Date and Time of Last Access
Date and Time of Last Write
File Size
Allocated File Size
File Attributes

- ✓ This is referred to as Level 1 File Information..
- ✓ The DIR command displays a subset of this information.
- ✓ An application may request Level 1 information with the DosQueryFileInfo or DosQueryPathInfo functions.

20FILES15 920808

Extended File Attributes

- ✓ Extended Attributes (EAs) may be used to describe a file to another application, the operating system or the file system.
- ✓ EAs may be used for such things as:
 - Store notes such as the creator of a file
 - Categorize files such as source, icons, bit maps, etc.
 - Describe the format of data in a file
 - Append additional data to a file
- ✓ EAs are not part of the file data but, are maintained by the file system.
- ✓ More than one EA may be associated with a file.
- ✓ Each EA consist of an ASCII Name and a Value of any data type.
- ✓ The maximum size of each EA is 64Kb.
- ✓ EAs are classified as critical or non-critical.

20FILE16 920808

Naming Extended Attributes

- ✓ Application specific EAs should be prefixed with the company name and product name.

Company - IBM

Product - DisplayWrite

Attribute - Stuff

EA Name = IBM_DW.Stuff

- ✓ A set of Standard Extended Attributes (SEAs) define common info associated with a file.

- ✓ SEA names are prefixed with a period (.).

.TYPE
.KEYPHRASES
.SUBJECT
.COMMENTS
.HISTORY
.VERSION
.LONGNAME
.ICON
.ASSOCTABLE
.CODEPAGE

- ✓ Reserved prefixes are \$ @ & +

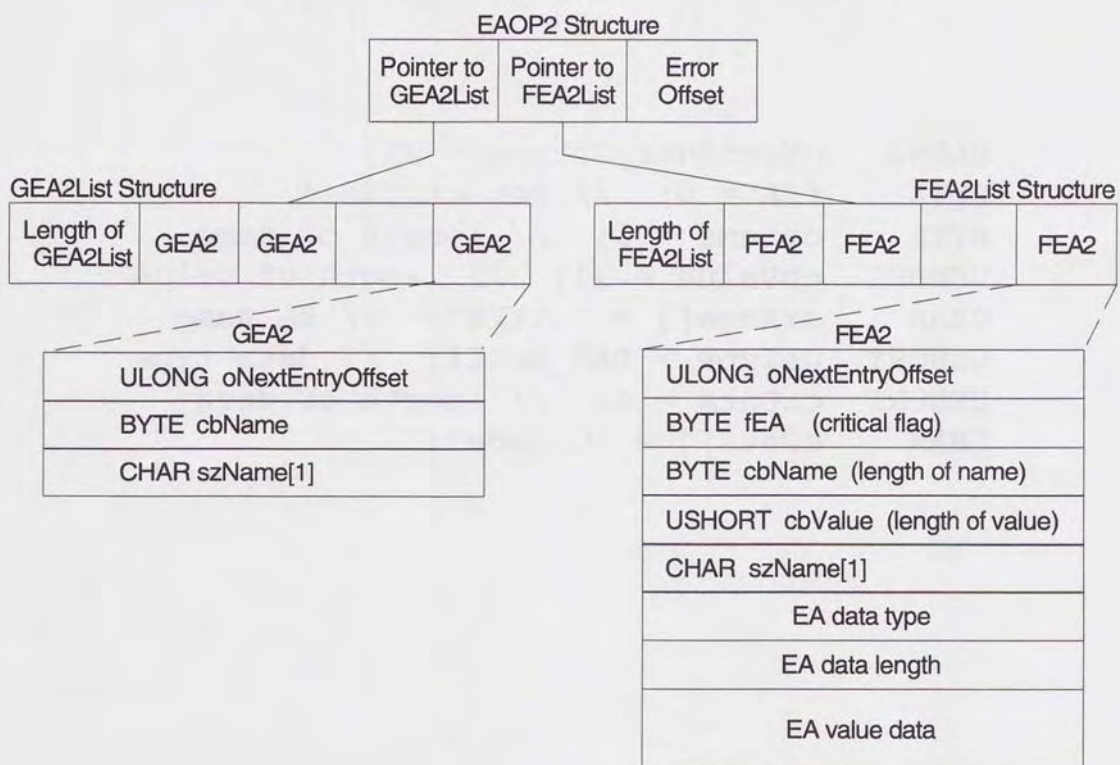
20FILE17 920808

Extended Attribute Data Types

Description	Data Type	Constant
Length-preceded binary	FFFE	EAT_BINARY
Length-preceded ASCII	FFFD	EAT_ASCII
Length-preceded bitmap	FFFB	EAT_BITMAP
Length-preceded metafile	FFFA	EAT_METAFILE
Length-preceded icon	FFF9	EAT_ICON
ASCII EA name of associated data	FFEE	EAT_EA
Multi-valued, multi-typed data	FFDF	EAT_MVMT
Multi-valued, single-typed data	FFDE	EAT_MVST
ASN.1 field	FFDD	EAT_ASN1
Reserved	8000- FFDC	
User-definable	0000- 7FFF	

- ✓ The Value of an EA may be any type of data.
- ✓ The first word of the EA data should indicate the data type.

Extended Attribute Data Structures



20FILE19

Example of a Single-Value EA

```
ULONG    oNextEntryOffset = 25;
BYTE      fEA = 0; // non-critical
BYTE      cbName = 5; // length of name
USHORT    cbValue = 11; // length of value
CHAR      szName[] = ".TYPE"; // EA name
USHORT    usType = EAT_ASCII; // data type
USHORT    cbData = 6; // length of data
CHAR      cData[] = "C Code";
```


Example of a Multi-Value, Single-Type EA

```
ULONG    oNextEntryOffset = 56;
BYTE     fEA = 0; // non-critical
BYTE     cbName = 16; // length of name
USHORT   cbValue = 31; // length of value
CHAR     szName[] = "PCRI_SAMPLE.MVST"; // EA name
USHORT   usType = EAT_MVST;
USHORT   usCodePage = 0; // use default
USHORT   usNumEntries = 3; // number of entries
USHORT   usDataType = EAT_ASCII;
USHORT   cbData1 = 5; // length of 1st entry
CHAR     cData1[] = "Hello";
USHORT   cbData2 = 4;
CHAR     cData2[] = "OS/2";
USHORT   cbData3 = 5;
CHAR     cData3[] = "World";
```

20FILES21 930211

Example of a Multi-Value, Multi-Type EA

```
ULONG    oNextEntryOffset = 55;
BYTE     fEA = 0; // non-critical
BYTE     cbName = 16; // length of name
USHORT   cbValue = 30; // length of value
CHAR     szName[] = "PCRI_SAMPLE.MVST"; // EA name
USHORT   usType = EAT_MVMT;
USHORT   usCodePage = 0; // use default
USHORT   usNumEntries = 2; // number of entries
USHORT   usDataType = EAT_ASCII;
USHORT   cbData1 = 11; // length of 1st entry
CHAR     cData1[] = "Hello World";
USHORT   usDataType = EAT_BINARY;
USHORT   cbData2 = 4;
BYTE     bData2[4] = {0x68, 0x65, 0x6c, 0x6f};
```

Preserving EAs

- ✓ Non-EA aware programs may not open files with CRITICAL EAs except for replacement.
- ✓ Files with non-critical EAs may be accessed by non-EA aware programs.
- ✓ EA-aware programs do not have any restrictions.
- ✓ EAs may be lost when files are:
 - Transferred to a down-level system
 - Opened for replacement by non-EA aware programs
 - Sent over comm lines
- ✓ The EAUTIL program allows EAs to be handled separately from the associated file.
- ✓ EAs may be stripped and placed in a separate file and then reattached to the associated file.

20FILE23 920808

Managing EAs

- ④ EAs may be defined when a file/subdirectory is created with DosOpen/DosCreateDir.
- ④ EAs may be defined for existing files with DosSetFileInfo or DosSetPathInfo.
- ④ EAs may be examined with DosQueryFileInfo, DosQueryPathInfo or DosEnumAttribute.
- ④ Files with specific EAs may be searched with DosFindFirst and DosFindNext.

Queues

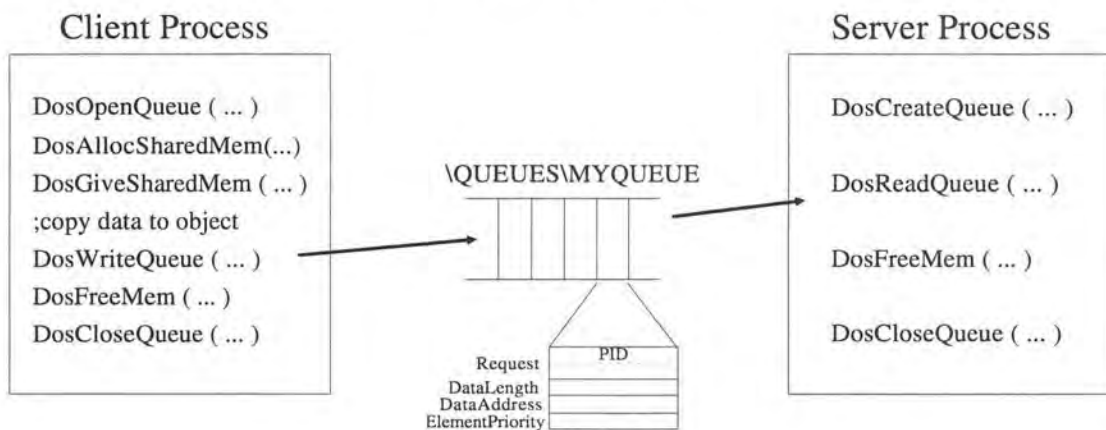
Creating/Peeking/Reading Queues From A Server Process

Opening/Writing/Closing Queues From A Client Process

Monitoring Multiple Queues From A Server Process

20QUES01 920810

Queue Overview



- ✓ Only the queue creator may read the queue.
- ✓ The Client process allocates the shared object and passes the pointer and size to the Server process through the queue.
- ✓ You could also use 'named' shared objects and the suballocation API's instead of DosGiveSharedMem.

DosCreateQueue

DosCreateQueue (PHQUEUE phq, -> r/w queue handle
 ULONG priority, <- QUE_constants
 PSZ pszName); <- \QUEUES\...

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
332	ERROR_QUEUE_DUPLICATE
334	ERROR_QUEUE_NO_MEMORY
335	ERROR_QUEUE_INVALID_NAME

0L	QUE_FIFO
1L	QUE_LIFO
2L	QUE_PRIORITY
4L	QUE_CONVERT_ADDRESS

✓ The process creating a queue is the server.

✓ Processes opening the queue are clients.

```
HQUEUE hq;  
DosCreateQueue ( &hq, QUE_LIFO, "\\queues\\MyQue" );
```

20QUES03 931119

DosOpenQueue

DosOpenQueue (PPID ppid, -> pid of server
PHQUEUE phq, -> write queue handle
PSZ pszName); <- \QUEUES\...

0	NO_ERROR
334	ERROR_QUE_NO_MEMORY
341	ERROR_QUE_PROC_NO_ACCESS
343	ERROR_QUE_NAME_NOT_EXIST

☑ Enables access to a previously created queue.

```
DosOpenQueue ( &pid, &hq, "\\queues\\MyQue" );
```

20QUES04 921021

* you need the PID of server in order to
set up givable memory to share with
server.

DosWriteQueue

DosWriteQueue (HQUEUE hq, <- queue handle
 ULONG request, <- 32-bit application specific
 ULONG cbData, <- data size
 PVOID pbData, <- data pointer
 ULONG priority); <- priority value, 15=high, 0=low

0 NO_ERROR
334 ERROR_QUE_NO_MEMORY
337 ERROR_QUE_INVALID_HANDLE

PID
request
cbData
pbData
priority

```
DosWriteQueue ( hq, 0x1ff, 2000, pBuffer, 0 );
```

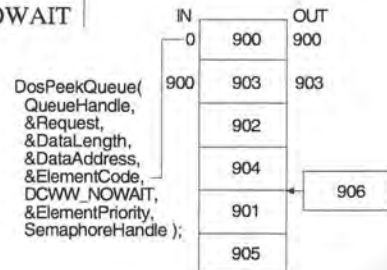
DosPeekQueue

DosPeekQueue (HQUEUE hq, <- queue handle
 PREQUESTDATA pRequest,-> clients pid and request
 PULONG pcbData, -> data size
 PPVOID ppbuf, -> data address
 PULONG element, <-> 0 input, peek 1st element
 N output, specific element
 BOOL32 nowait, <- DCWW_ parameter
 PBYTE ppriority, -> priority value
 HEV hsem); <- event semaphore handle

```
0 NO_ERROR
87 ERROR_INVALID_PARAMETER
330 ERROR_QUEUE_PROC_NOT_OWNED
333 ERROR_QUEUE_ELEMENT_NOT_EXIST
337 ERROR_QUEUE_INVALID_HANDLE
342 ERROR_QUEUE_EMPTY
433 ERROR_QUEUE_INVALID_WAIT
```

```
typedef struct _REQUESTDATA
{
  PID pid;
  ULONG ulData;
} REQUESTDATA, *PREQUESTDATA;
```

```
0 DCWW_WAIT
1 DCWW_NOWAIT
```



20QUES06 921027

DosReadQueue

DosReadQueue (HQUEUE hq, <- queue handle
 PREQUESTDATA pRequest, -> clients pid and request
 PULONG pcbData, -> data size
 PPVOID ppbuf, -> data address
 ULONG element, <- element code
 BOOL32 wait, <- DCWW_ parameter
 PBYTE ppriority, -> priority value
 HEV hsem); <- event semaphore handle

```

0 NO_ERROR
87 ERROR_INVALID_PARAMETER
330 ERROR_QUEUE_PROC_NOT_OWNED
333 ERROR_QUEUE_ELEMENT_NOT_EXIST
337 ERROR_QUEUE_INVALID_HANDLE
342 ERROR_QUEUE_EMPTY
433 ERROR_QUEUE_INVALID_WAIT
    
```

```

typedef struct _REQUESTDATA
{
    PID pid;
    ULONG ulData;
} REQUESTDATA, *PREQUESTDATA;
    
```

```

0 DCWW_WAIT
1 DCWW_NOWAIT
    
```

```

DosReadQueue ( hq, &Request, &cbData, &pbuff, 0, DCWW_NOWAIT, &pri, hev );
    
```

20QUES07 921027

DosQueryQueue, DosPurgeQueue, DosCloseQueue

DosQueryQueue (HQUEUE hq, <- queue handle
 PULONG pcbEntries); -> number of queue elements

0 NO_ERROR
337 ERROR_QUE_INVALID_HANDLE

DosPurgeQueue (HQUEUE hq); <- queue handle

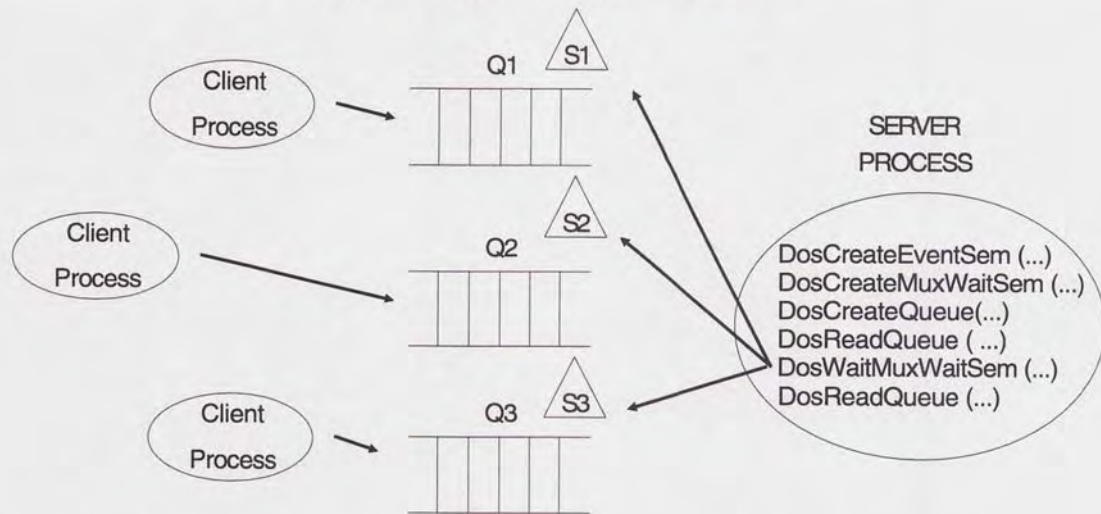
0 NO_ERROR
330 ERROR_QUE_PROC_NOT_OWNED
337 ERROR_QUE_INVALID_HANDLE

DosCloseQueue (HQUEUE hq); <- queue handle

0 NO_ERROR
337 ERROR_QUE_INVALID_HANDLE

20QUES08 920810

Monitoring Multiple Queues



- ✓ Allows the monitoring of multiple queues by the server process.
- ✓ When a client writes to the queue, the event semaphore will automatically be opened and posted.
- ✓ The server could query the event semaphore to determine the number of posts issued.

20ques09 920810

Notes

Dynamic Link Libraries

Static Linking

Dynamic Linking

Building a Dynamic Link Library

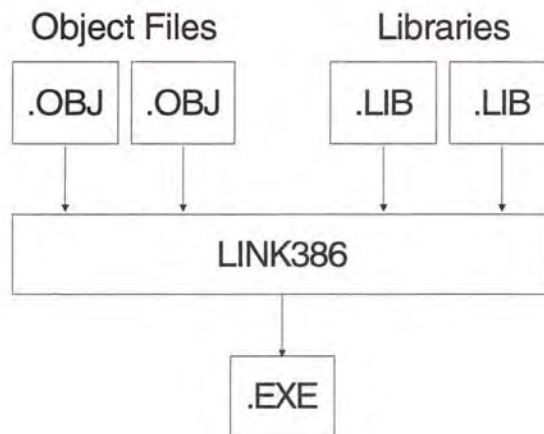
Dynamic Linking at Run Time

DLL Initialization

20DLLS01 920814

SEE DBL.4 in supplemental handbook

Static Linking



- ✓ All object file and library file code and data are in the .EXE even if not all will be used.

Disadvantages of Static Linking

- ✓ All of the target code and data must be available at link time.
- ✓ All modules must be re-linked if the target code changes.
- ✓ The target code can not be shared among applications.
- ✓ The operating system may have multiple copies of the same code/data in memory and on the disk.
- ✓ The executable (.EXE) files are large.

20DLLS02 920811

Advantages and Disadvantages of Dynamic Link Libraries

✓ Advantages:

OS/2 only keeps one copy of the code in memory.

Upgrades are easier; simply replace the DLL.

Dynlink procedures can call other dynlink procedures.

.EXE files are smaller.

✓ Disadvantage:

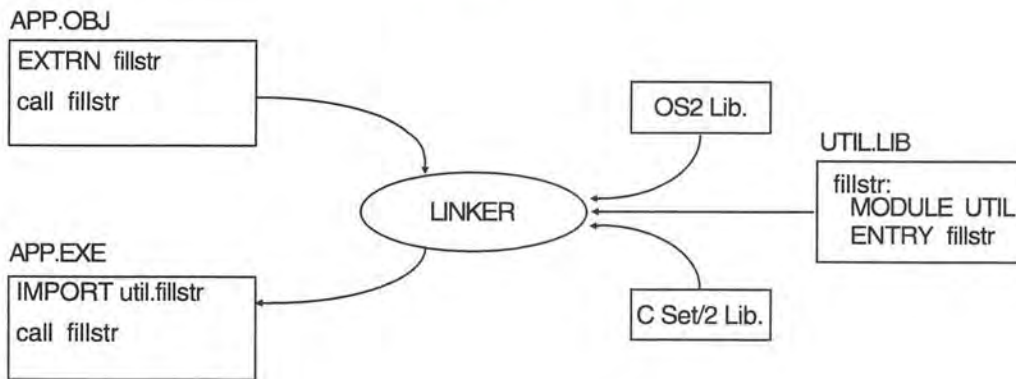
Program loading may take longer.

Dynamic Links vs Processes

- ④ OS/2 views dynlinks as subroutines, not as separate processes.
- ④ The dynlink procedure runs in the context of its calling thread and process .
- ④ Dynlink calls are fast since there is no change in the privilege level.
- ④ Dynlink objects are mapped to the same linear addresses for all processes.

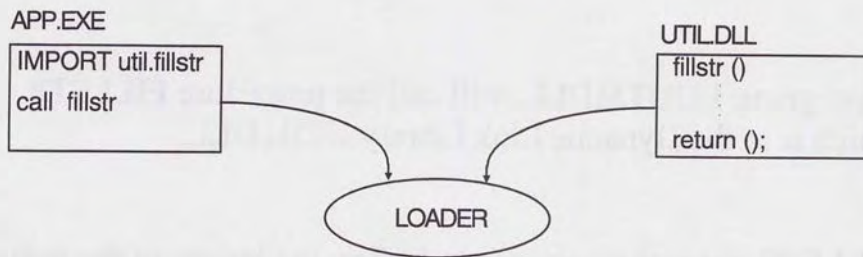
20DLLS05 930910

Dynamic Linking



- ✓ The program makes an external reference to a procedure:
EXTERN USHORT APIENTRY fillstr (PSZ pszBuff, USHORT usLength, CHAR chFill);
- ✓ The object file is linked to a special library file that contains the module name where the procedure 'fillstr' lives (UTIL).
- ✓ The APP.EXE file does NOT contain the target Code or Data.

Load Time



- ✓ The OS/2 loader brings in the target code and fixes up the call.
- ✓ 'fillstr' runs as a near procedure.
- ✓ The Dynamic Link Library file (DLL) contains the code for 'fillstr'.

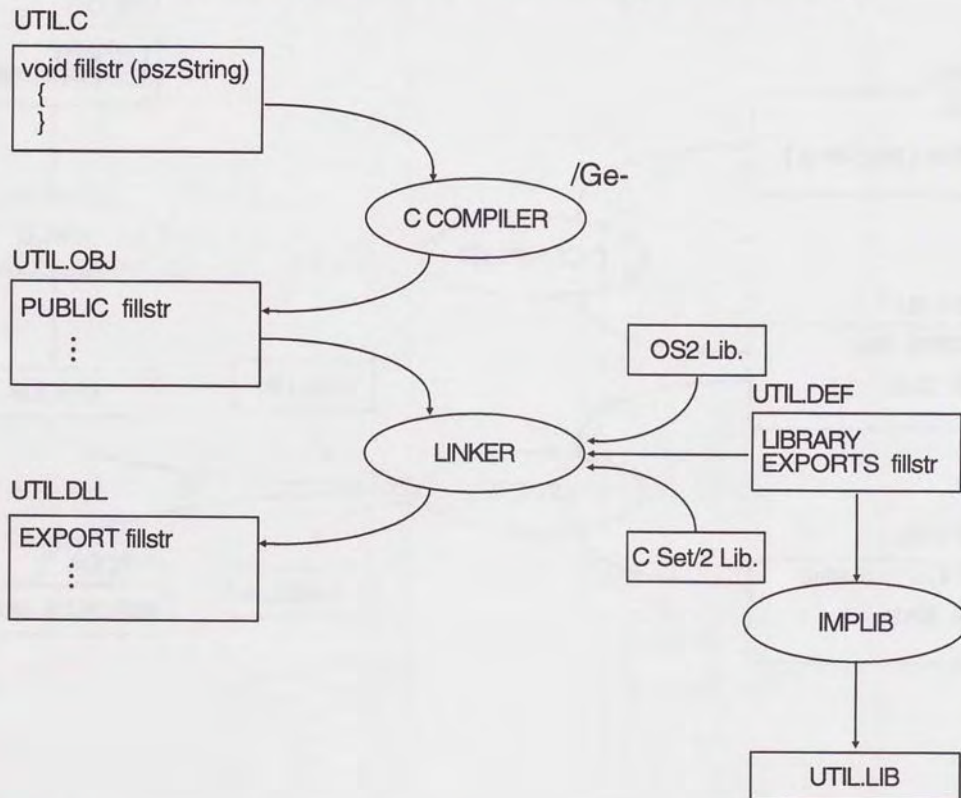
20DLLS07 920814

Building a DLL

- ④ A program, LODTMDLL, will call the procedure FILLSTR which is in the Dynamic Link Library UTIL.DLL.
- ④ FILLSTR expects a pointer to a buffer, the length of the buffer and the character with which to fill the buffer.

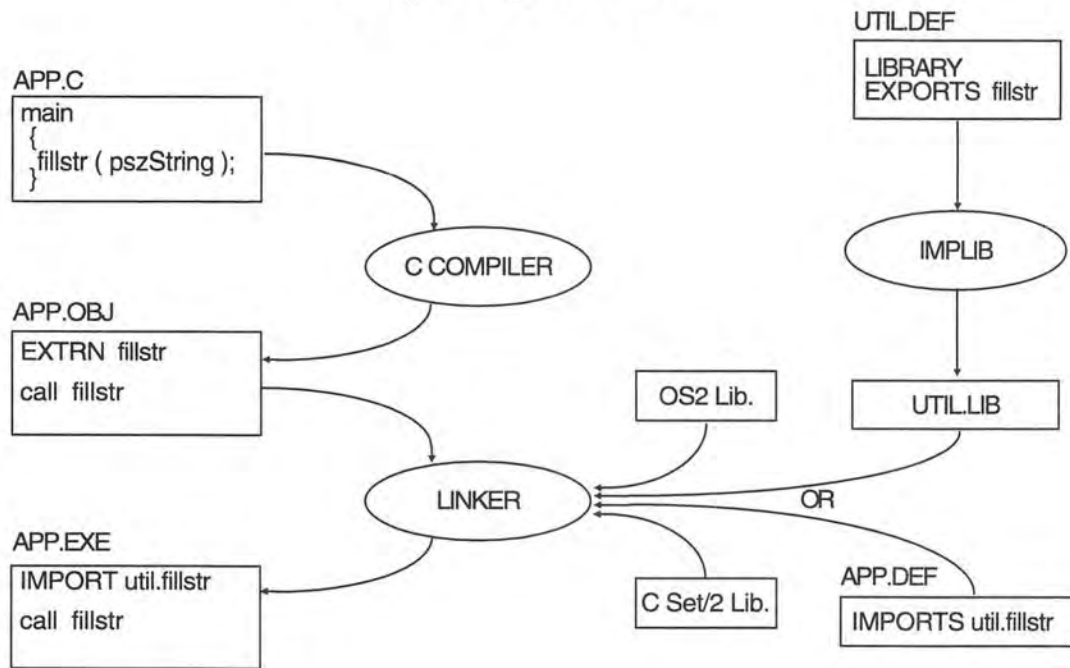
20DLLS08 920812

Preparing the .DLL and Import Library



20DLLS09 930325

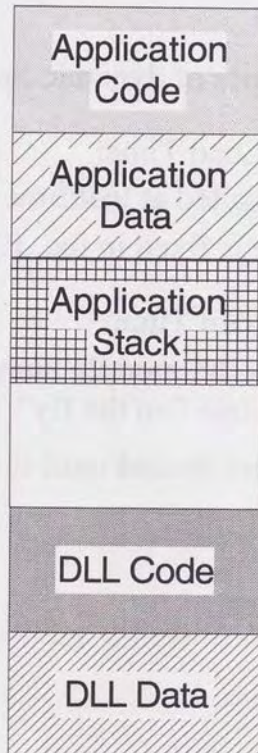
Preparing the .EXE



20DLL10 940125

Variables in Main and in the DLL

```
CHAR a;  
main (void)  
  CHAR b;  
  static CHAR c;  
  
dllsub ( a, b, c );
```



```
CHAR dlldata[28];  
dllsub ( CHAR a, CHAR b, CHAR c)  
{  
  CHAR dlla;  
  a = 'x';  
}
```

20DLLS15 920925

Methods of Dynamic Linking

- ✓ OS/2 supports two types of dynamic linking.
- ✓ Dynamic Linking at Load Time:
 - Target code is loaded at loadtime.
 - Procedure name is fixed in the .EXE file.
- ✓ Dynamic Linking at Run Time:
 - Program creates the module name and the procedure name "on the fly".
 - Target code is not loaded until it is needed.

Dynamic Linking at Run Time

- ① The process creates the module and procedure name.
- ② The process then calls `DosLoadModule` to find and load the target DLL module.
- ③ A call to `DosQueryProcAddr` returns a pointer to the target procedure.
- ④ The procedure is invoked directly through the pointer.
- ⑤ When done, the process calls `DosFreeModule` to release the DLL module.

20DLLS17 921008

DosLoadModule

DosLoadModule (PSZ pszName, --> ptr to msg. buffer
 ULONG cbName, <-- size of msg. buffer
 PSZ pszModname, <-- name of module to load
 PHMODULE phmod); --> ptr to returned module handle

0	NO_ERROR	127	ERROR_PROC_NOT_FOUND
2	ERROR_FILE_NOT_FOUND	180	ERROR_INVALID_SEGMENT_NUMBER
3	ERROR_PATH_NOT_FOUND	182	ERROR_INVALID_ORDINAL
4	ERROR_TOO_MANY_OPEN_FILES	190	ERROR_INVALID_MODULETYPE
5	ERROR_ACCESS_DENIED	191	ERROR_INVALID_EXE_SIGNATURE
8	ERROR_NOT_ENOUGH_MEMORY	192	ERROR_EXE_MARKED_INVALID
11	ERROR_BAD_FORMAT	194	ERROR_ITERATED_DATA_EXCEEDS_64K
26	ERROR_NOT_DOS_DISK	195	ERROR_INVALID_MINALLOCSIZE
32	ERROR_SHARING_VIOLATION	196	ERROR_DYNLINK_FROM_INVALID_RING
33	ERROR_LOCK_VIOLATION	198	ERROR_INVALID_SEGDPL
36	ERROR_SHARING_BUFFER_EXCEEDED	199	ERROR_AUTODATASEG_EXCEEDS_64K
95	ERROR_INTERRUPT	201	ERROR_RELOCSR_CHAIN_EXCEEDS_SEGLIMIT
108	ERROR_DRIVE_LOCKED	206	ERROR_FILENAME_EXCED_RANGE
123	ERROR_INVALID_NAME	295	ERROR_INIT_ROUTINE_FAILED

```
DosLoadModule ( &Msgbuf, 254, "MYDLL", &hmod );
```

DosQueryProcAddr, DosFreeModule

DosQueryProcAddr (HMODULE hmod, <-- module handle
 ULONG ordinal, <-- ordinal number
 PSZ pszName, <-- procedure name
 PFN *ppfn); --> ptr to returned proc addr

0 NO_ERROR
6 ERROR_INVALID_HANDLE
123 ERROR_INVALID_NAME
65079 ERROR_ENTRY_IS_CALLGATE

void (*pfnReport) (PSZ pszMsg, APIRET rc, PSZ MyName);

rc = DosQueryProcAddr (hmod, 0, "report", (PFN *) &pfnReport);
if (rc) exit (1);
pfnReport ("QueryProcAddress successful!\n", 0, "INFOMSG: ");

DosFreeModule (HMODULE hmod); <-- module handle

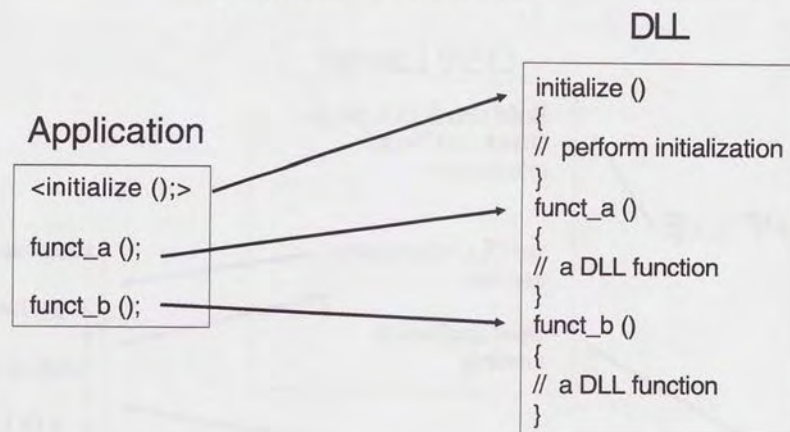
0 NO_ERROR
6 ERROR_INVALID_HANDLE
12 ERROR_INVALID_ACCESS
95 ERROR_INTERRUPT

20DLLS19 940119

Data Objects in DLLs

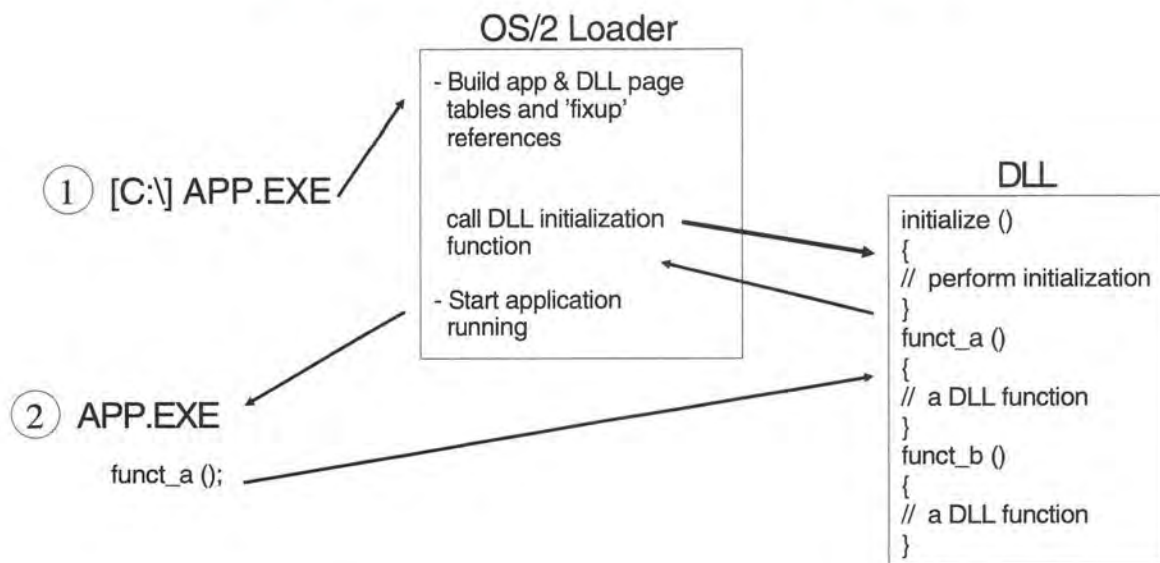
- ④ Regular .EXE programs get unique copies of all data objects whenever the .EXE file is loaded (default).
- ④ DLL modules do NOT. The loader will only load one copy of the data objects for DLL modules by default.
- ④ This may be overridden with an entry in the module definition file:
DATA MULTIPLE NONSHARED
- ④ DLL's use the stack memory of the calling process.

DLL Initialization



- ✓ A DLL may need to perform initialization functions before the application invokes any procedures in the DLL.
- ✓ The initialization may involve:
 - allocating memory
 - creating or opening files
 - setting up devices

Flow of Control for DLL Initialization



- ✓ The loader calls the DLL's initialization function before the application can call other DLL functions. This will also occur during `DosLoadModule`.

_DLL_InitTerm

unsigned long _DLL_InitTerm (unsigned long modhandle, --> DLL file handle
unsigned long flag); --> 0=init, 1=term

```
#pragma linkage ( _DLL_InitTerm, system)
int _CRT_init (void);
unsigned long _DLL_InitTerm( unsigned long modhandle
                             unsigned long flag )
{
    switch( flag )
    {
        case 0:
            if ( _CRT_init( ) == -1 ) return 0;
            printf ("printf now works in _DLL_InitTerm(0) \n")
            break;
        case 1:
            _CRT_term ();
            break;
    }
    return 1; /* non-zero value returned to continue load */
}
```

- ✓ IBM C Set/2 provides a default procedure, _DLL_InitTerm, which initializes and terminates the C run-time environment.
- ✓ The user may optionally provide a modified routine if additional initialization or termination actions are required by the user DLL.

20DLLS23 930206

Types of DLL Initialization

DLL's Module Definition File

LIBRARY	[UTIL]	[INITGLOBAL* INITINSTANCE]	[TERMGLOBAL* TERMINSTANCE]
---------	--------	-------------------------------	-------------------------------

* default

INITGLOBAL Loader calls the init. routine when the 1st process attaches to the DLL. This method is good if the DLL shares resources with all processes.

INITINSTANCE Loader calls the init. routine each time a new process attaches to the DLL. This method allocates a new set of resources for each process. Implies **TERMINSTANCE**.

- ④ **INITINSTANCE** may be combined with 'exit lists' to allocate and deallocate resources on a per process basis.

Exception Management

Asynchronous/Synchronous Exceptions

Signal Exceptions

Registering Exception Handlers

Exception Management Structures

20XCPT01 920817

System Exceptions

- ✓ An exception is an abnormal condition that can occur during program execution.
- ✓ Common causes of exceptions include:
 - I/O errors
 - Protection violations
 - Math errors
 - Intervention by the user or by another process
- ✓ Activities that can cause exceptions include:
 - Invalid memory access
 - Dividing by zero
 - Pressing Ctrl+Break

Types of Exceptions

- ✓ Asynchronous Exceptions are caused by events that are external to the execution of the thread.
- ✓ An asynchronous exception may be caused by the user pressing Ctrl+Break or Ctrl+C, or by a process calling DosKillProcess.
- ✓ Synchronous exceptions are caused by events that are internal to the execution of the thread.
- ✓ Synchronous exceptions may be caused by:
 - Invalid memory access
 - Attempt to execute an illegal instruction
 - Math calculation errors
 - Illegal stack operation

20XCPT04 920817

Signal Exceptions

- ✓ Signal exceptions are asynchronous events sent to a thread when the user presses certain keys or when another thread or process initiates the exception.
- ✓ There are three types of signal exceptions:
 - XCPT_SIGNAL_BREAK - Ctrl+Break
 - XCPT_SIGNAL_INTR - Ctrl+C
 - XCPT_SIGNAL_KILLPROC - DosKillProcess
- ✓ All three signals are delivered by a single exception, XCPT_SIGNAL.
- ✓ The exception handler for thread 1 can handle none, some or all of the signals
- ✓ In order for a process to receive signal exceptions when multiple processes are running in the same screen group, the process must request signal exception focus (DosSetSignalExceptionFocus).
- ✓ A process can send the XCPT_SIGNAL exception to another process (DosSendSignalException).

20XCPT05 920817

Raising Exceptions

- ④ Asynchronous exceptions that have been deferred in a must-complete section are dispatched automatically by the system when the section is exited.
- ④ A synchronous exception that has been deferred must be raised by `DosRaiseException`.
- ④ `DosRaiseException` can also be used to simulate either a synchronous or an asynchronous exception.

20XCPT06 921027

Error Pop-Up Screens

- ④ Some error conditions, such as general protection violations, cause the operating system to display a pop-up screen containing information about the error.
- ④ The application can disable error pop-up screens.
- ④ DosError can disable or enable both exception and hard error pop-ups.

20XCPT07 920817

Exception Handlers

- ✓ The system provides a default exception handler which, in most cases, will terminate the application.
- ✓ The application can register its own subroutine to process exceptions, allowing it to avoid termination (DosSetExceptionHandler).
- ✓ If multiple exception handlers are registered, they are invoked in Last-In-First-Out order.
- ✓ If a process termination exception occurs and exit-list procedures have been registered, the exit-list procedure will get control after the exception handler.

20XCPT03 930910

Exception Management Data Structures

- ✓ **ExceptionRegistrationRecord**
Used to establish an exception handler.
- ✓ **ExceptionReportRecord**
Describes an exception.
- ✓ **ContextRecord**
Describes the machine state at the time of an exception.
- ✓ **DispatcherContext**
Receives information on nested exceptions and collided unwinds.

20XCPT08 920817

ExceptionRegistrationRecord

```
typedef struct _EXCEPTIONREGISTRATIONRECORD
{
    struct _EXCEPTIONREGISTRATIONRECORD * volatile prev_structure; // ptr to previous handler
    _err * volatile ExceptionHandler;                               // ptr to handler being registered
} EXCEPTIONREGISTRATIONRECORD,
* PEXCEPTIONREGISTRATIONRECORD
```

- ✓ When a procedure begins, it must create an ExceptionRegistrationRecord on the stack, fill in the pointer to the exception handler routine, and link to the front of the exception handler chain by calling DosSetExceptionHandler.
- ✓ When the procedure ends, it must remove the ExceptionRegistrationRecord from the chain by calling DosUnsetExceptionHandler.

20XCPT09 920817

ExceptionReportRecord

```
typedef struct _EXCEPTIONREPORTRECORD
```

```
{
    ULONG ExceptionNum;           // exception number
    ULONG fHandlerFlags;         // nesting/unwinding flags
    struct _EXCEPTIONREPORTRECORD *NestedExceptionReportRecord; // ptr to nested ExceptionReportRecord
    PVOID ExceptionAddress;      // address where exception occurred
    ULONG cParameters;          // number of ExceptionInfo elements
    ULONG ExceptionInfo[4];      // exception specific info
} EXCEPTIONREPORTRECORD, *PEXCEPTIONREPORTRECORD;
```

0x0	XCPT_UNKNOWN_ACCESS
0x1	XCPT_READ_ACCESS
0x2	XCPT_WRITE_ACCESS
0x4	XCPT_EXECUTE_ACCESS
0x8	XCPT_SPACE_ACCESS
0x10	XCPT_LIMIT_ACCESS

0x1	EH_NONCONTINUABLE
0x2	EH_UNWINDING
0x4	EH_EXIT_UNWIND
0x8	EH_STACK_INVALID
0x10	EH_NESTED_CALL

0x80000001	XCPT_GUARD_PAGE_VIOLATION	0xC0000097	XCPT_FLOAT_INVALID_OPERATION
0x80010001	XCPT_UNABLE_TO_GROW_STACK	0xC0000098	XCPT_FLOAT_OVERFLOW
0xC0000005	XCPT_ACCESS_VIOLATION	0xC0000099	XCPT_FLOAT_STACK_CHECK
0xC0000006	XCPT_IN_PAGE_ERROR	0xC000009A	XCPT_FLOAT_UNDERFLOW
0xC000001C	XCPT_ILLEGAL_INSTRUCTION	0xC000009B	XCPT_INTEGER_DIVIDE_BY_ZERO
0xC000001D	XCPT_INVALID_LOCK_SEQUENCE	0xC000009C	XCPT_INTEGER_OVERFLOW
0xC0000024	XCPT_NONCONTINUABLE_EXCEPTION	0xC000009D	XCPT_PRIVILEGED_INSTRUCTION
0xC0000025	XCPT_INVALID_DISPOSITION	0xC000009E	XCPT_DATATYPE_MISALIGNMENT
0xC0000026	XCPT_UNWIND	0xC000009F	XCPT_BREAKPOINT
0xC0000027	XCPT_BAD_STACK	0xC00000A0	XCPT_SINGLE_STEP
0xC0000028	XCPT_INVALID_UNWIND_TARGET	0xC0010001	XCPT_PROCESS_TERMINATE
0xC0000093	XCPT_ARRAY_BOUNDS_EXCEEDED	0xC0010002	XCPT_ASYNC_PROCESS_TERMINATE
0xC0000094	XCPT_FLOAT_DENORMAL_OPERAND	0xC0010003	XCPT_SIGNAL
0xC0000095	XCPT_FLOAT_DIVIDE_BY_ZERO	0xC0010004	XCPT_B1NPX_ERRATA_02
0xC0000096	XCPT_FLOAT_INEXACT_RESULT		

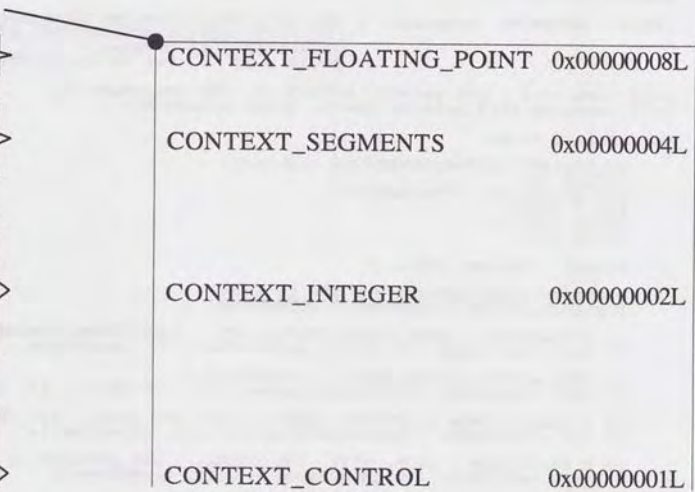
20XCPT10 921009

ContextRecord

```
typedef struct _CONTEXT
```

```
{
    ULONG ContextFlags;
    ULONG ctx_env[7];
    FPREG ctx_stack[8];
    ULONG ctx_SegGs;
    ULONG ctx_SegFs;
    ULONG ctx_SegEs;
    ULONG ctx_SegDs;
    ULONG ctx_RegEdi;
    ULONG ctx_RegEsi;
    ULONG ctx_RegEax;
    ULONG ctx_RegEbx;
    ULONG ctx_RegEcx;
    ULONG ctx_RegEdx;
    ULONG ctx_RegEbp;
    ULONG ctx_RegEip;
    ULONG ctx_SegCs;
    ULONG ctx_EFlags;
    ULONG ctx_RegEsp;
    ULONG ctx_SegSs;
}
```

```
} CONTEXTRECORD, *PCONTEXTRECORD;
```



CONTEXT_FLOATING_POINT	0x00000008L
CONTEXT_SEGMENTS	0x00000004L
CONTEXT_INTEGER	0x00000002L
CONTEXT_CONTROL	0x00000001L

Exception Handling Example

```
// pagefault.c      example of page fault exception handler

#define INCL_DOS
#define INCL_DOSERRORS
#define INCL_DOSEXCEPTIONS
#include <os2.h>
#include <stdio.h>

#define STDOUT 1
#define STDERR 2

APIRET APIENTRY MyHandler ( PEXCEPTIONREPORTRECORD pERepRec,
                           PEXCEPTIONREGISTRATIONRECORD pERegRec,
                           PCONTEXTRECORD pCtxRec,
                           PVOID p ); // ptr to dispatcher context

void terminate ( PSZ pszMsg, APIRET rc, PSZ pszMyName );
void querymem (PCH pchStartQuery, ULONG ulLength);

void main ( void)
{
    EXCEPTIONREGISTRATIONRECORD xcpthand;
    APIRET rc;
    PSZ pszMyName = "PAGEFALT:";
    PCH pBuf;
    ULONG ulAttr;
    ULONG ulX;

    setbuf ( stdout, NULL );

    xcpthand.prev structure = END_OF_CHAIN;
    xcpthand.ExceptionHandler = MyHandler;

    rc = DosError ( FERR_DISABLEEXCEPTION | FERR_DISABLEHARDERR ); // disable pop-ups
    if (rc) terminate ("DosError problem\n", rc, pszMyName);

    rc = DosSetExceptionHandler ( &xcpthand );
    if (rc) terminate ("SetExceptionHandler problem\n", rc, pszMyName);

    rc = DosAllocMem ( (PVOID) &pBuf, 8192, PAG_READ | PAG_WRITE );
    if (rc) terminate ("AllocMem error\n", rc, pszMyName);

    rc = DosSetMem ( pBuf, 4096, PAG_COMMIT | PAG_DEFAULT );
    if (rc) terminate ("SetMem error\n", rc, pszMyName);

    for (ulX = 4090; ulX < 4100; ulX++)
    {
        pBuf[ulX] = '1';
        printf ("ulX = %d pBuf[ulX] = %x\n", ulX, pBuf[ulX] );
    }

    rc = DosUnsetExceptionHandler ( &xcpthand );
    if (rc) terminate ("UnsetExceptionHandler problem\n", rc, pszMyName);

    printf ("Terminating\n");

    DosExit ( EXIT_PROCESS, NO_ERROR );
}
```

20XCPT12921009

Exception Handling Example cont'd

```
APIRET WINAPI MyHandler ( PEXCEPTIONREPORTRECORD pRepRec,
                          PEXCEPTIONREGISTRATIONRECORD pRegRec,
                          PCONTEXTRECORD pCtxRec,
                          PVOID p )
{
    ULONG cbWritten, ulMemSize, flMemAttrs;
    APIRET rc;

    // Access violation at a know location
    if ( pRepRec->ExceptionNum == XCPT_ACCESS_VIOLATION &&
        pRepRec->ExceptionAddress != (PVOID)-XCPT_DATA_UNKNOWN )
    {
        // page fault
        if ( ( pRepRec->ExceptionInfo[0] == XCPT_READ_ACCESS ||
              pRepRec->ExceptionInfo[0] == XCPT_WRITE_ACCESS ) &&
            pRepRec->ExceptionInfo[1] != XCPT_DATA_UNKNOWN )
        {
            DosWrite ( STDERR, "\r\nHANDLER: Page Fault\r\n", 24, &cbWritten );
        }
        // now query the memory to find out why we faulted
        ulMemSize = 1;
        DosQueryMem ( (PVOID) pRepRec->ExceptionInfo[1],
                      &ulMemSize, &flMemAttrs );

        // If the memory is free or committed, we have some other problem.
        // If it is not free or not committed, commit it.
        if ( !(flMemAttrs & ( PAG_FREE | PAG_COMMIT )) )
        {
            DosWrite ( STDERR, "\r\nHANDLER: Attempt to access \"\
uncommitted memory\r\n", 49, &cbWritten );

            rc = DosSetMem ( (PVOID) pRepRec->ExceptionInfo[1], 1,
                            PAG_DEFAULT | PAG_COMMIT );

            if ( rc )
            {
                DosWrite ( STDERR, "\r\nError committing memory\r\n",
                          27, &cbWritten );
                return ( XCPT_CONTINUE_SEARCH ); // not handled
            }
            else
            {
                return ( XCPT_CONTINUE_EXECUTION ); // handled
            }
        } // end if
    } // end if
    return ( XCPT_CONTINUE_SEARCH ); // not handled
}
```

20XCPT13 930619

Summary of Exception Related Functions

DosAcknowledgeSignalException

DosEnterMustComplete

DosExitMustComplete

DosRaiseException

DosSendSignalException

DosSetExceptionHandler

DosSetSignalExceptionFocus

DosUnsetExceptionHandler

DosUnwindException

Session Management

Session Resources

Session Management

Session Hierarchy

Session Startup

20SESN01 921009

Overview of Sessions, Processes, & Threads

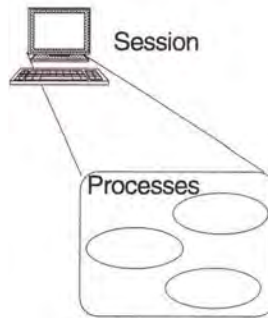
- ④ Session - A collection of one or more processes associated with a virtual console
- ④ Process - A collection of one or more threads and associated system resources.
- ④ Thread - A dispatchable unit of work
- ④ Resources are distributed between Sessions, Processes, and Threads.

Session Resources

- ✓ Video Buffer
- ✓ Keyboard Buffer
- ✓ Mouse Buffer
- ✓ Session ID
- ✓ Child Session

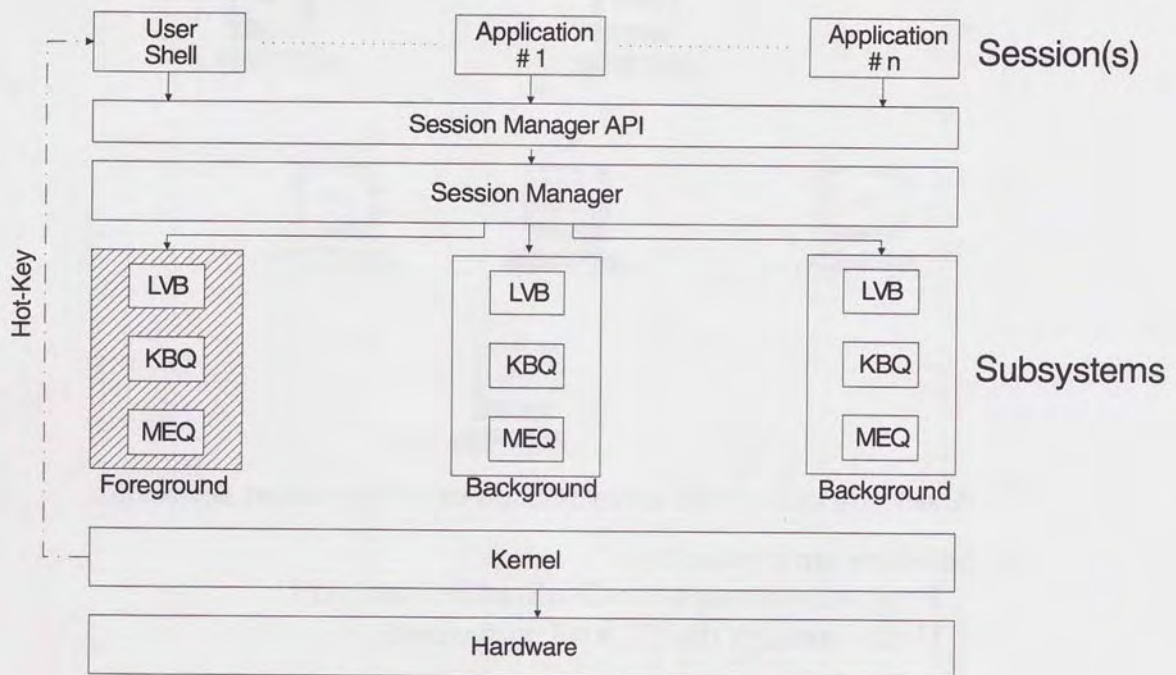
20SESN03 920814

Session



- ✓ A session is one or more processes that all share the same keyboard, mouse, and display.
- ✓ A maximum of 255 sessions is permitted. Those sessions may be any mixture of Full Screen, Graphic Window, Text Window or Virtual Dos Machines.
- ✓ Each session contains at least one process.

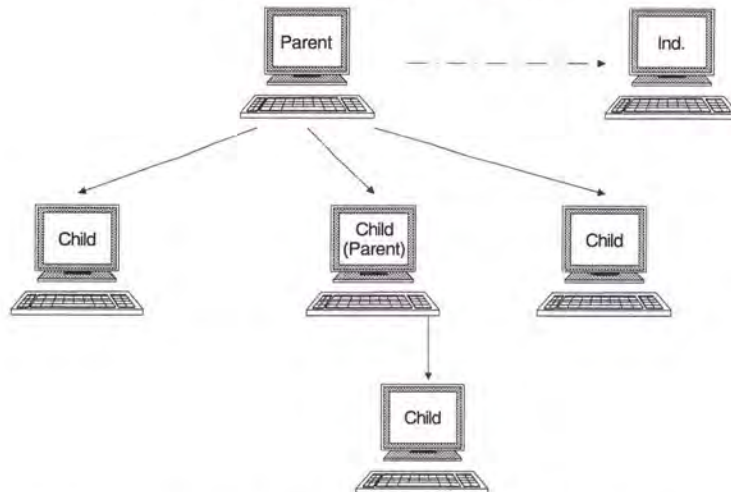
Session Management Components



- ✓ The Session Manager switches the 'focus' of the subsystem to the correct application.

20SEGN05 930210

Session Hierarchy



- ✓ A session can create other related or independent sessions.
- ✓ Sessions are created by:
 - Programs through the DosStartSession API
 - Users through the START command
- ✓ Each session has a session ID # (SESSID).
- ✓ If a child session is stopped, all descendants are stopped.

20SESN06 920814

Creating A Session

- ④ DosStartSession creates independent or dependent (child) sessions.
- ④ A Newly created session can be started in the foreground or background.
- ④ A Session ID and a process ID are returned to the Parent Session (valid only if Child Session).
- ④ The process in the new session can be the command processor .
- ④ A termination queue may be set up so that parent sessions can be notified when child sessions terminate.

20SESN07 921009

DosStartSession

DosStartSession (PSTARTDATA psd, <-- start session data
 PULONG pidSession, --> returned session id
 PPID ppid); --> returned process id

```

0 NO_ERROR
369 ERROR_SMG_INVALID_SESSION_ID
418 ERROR_SMG_INVALID_CALL
460 ERROR_SMG_PROCESS_NOT_PARENT
463 ERROR_SMG_RETRY_SUB_ALLOC
  
```

```
STARTDATA startdata;
```

```

startdata.Length = 24;
startdata.Related = SSF RELATED_CHILD;
startdata.FgBg = SSF FGEG_BACK;
startdata.TraceOpt = SSF TRACEOPT_NONE;
startdata.PgmTitle = "Brōwse";
startdata.PgmName = "c:\os20labs\browse.exe";
startdata.PgmInputs = "browse\0 main.c\0\0";
startdata.TermQ = 0;
  
```

```

DosStartSession ( &startdata, &pidSession,
                  &pid );
  
```

```

typedef struct _STARTDATA {
    USHORT Length;
    USHORT Related;
    USHORT FgBg;
    USHORT TraceOpt;
    PSZ PgmTitle;
    PSZ PgmName;
    PBYTE PgmInputs;
    PBYTE TermQ;
    PBYTE Environment;
    USHORT InheritOpt;
    USHORT SessionType;
    PSZ IconFile;
    ULONG PgmHandle;
    USHORT PgmControl;
    USHORT InitXPos;
    USHORT InitYPos;
    USHORT InitXSize;
    USHORT InitYSize;
    USHORT Reserved;
    PSZ ObjectBuffer;
    ULONG ObjectBuffLen;
} STARTDATA, *PSTARTDATA;
  
```

20SESIN08 921124

Setting Selectability and Bonding of a Child Session

- ✓ When a child session is selectable, the user can select it from the Window List or by using Alt+Esc.
- ✓ When a child session is nonselectable, the user CANNOT select the session.
- ✓ An application can establish a bond between a parent session and one of its child sessions.
- ✓ When the two sessions are bound, the operating system brings the child session to the foreground when the user selects the parent session.

20SESN09 920814

DosSetSession

DosSetSession (ULONG idSession, <-- session id
PSTATUSDATA psd); <-- ptr to status data structure

```
0 NO_ERROR
369 ERROR_MSG_INVALID_SESSION_ID
418 ERROR_MSG_INVALID_CALL
455 ERROR_MSG_INVALID_BOND_OPTION
456 ERROR_MSG_INVALID_SELECT_OPT
460 ERROR_MSG_PROCESS_NOT_PARENT
461 ERROR_MSG_INVALID_DATA_LENGTH
463 ERROR_MSG_RETRY_SUB_ALLOC
```

```
typedef struct _STATUSDATA {
    USHORT Length; /* 6 */
    USHORT SelectInd;
    USHORT BondInd;
} STATUSDATA, *PSTATUSDATA;
```

```
0 SET_SESSION_UNCHANGED
1 SET_SESSION_SELECTABLE
2 SET_SESSION_NON_SELECTABLE
1 SET_SESSION_BOND
2 SET_SESSION_NO_BOND
```

- ✓ This function may only be called by a parent session and only for a child session.
- ✓ The status of a session that was started as unrelated cannot be changed.
- ✓ When a bond is established with a child session, any bond that the parent had with another child session is broken.

```
STATUSDATA statusdata;
statusdata.length = 6
statusdata.Selected = SET_SESSION_UNCHANGED;
statusdata.BondInd = SET_SESSION_BOND;
DosSetSession ( idSession, &statusdata );
```

20SESN10 921027

DosSelectSession

DosSelectSession (ULONG idSession);

0 NO_ERROR
224 ERROR_SMG_NO_TARGET_WINDOW
369 ERROR_SMG_INVALID_SESSION_ID
418 ERROR_SMG_INVALID_CALL
459 ERROR_SMG_BAD_RESERVE
460 ERROR_SMG_PROCESS_NOT_PARENT
463 ERROR_SMG_RETRY_SUB_ALLOC

- ✓ A parent can force itself or any child session to the foreground.
- ✓ DosSelectSession only works if parent or any child is already in the foreground.
- ✓ If a child terminates while in the foreground, the parent becomes the foreground session.
- ✓ The PARENT HAS CONTROL!

DosStopSession

`DosStopSession (ULONG scope, ULONG idSession);`

0 NO_ERROR
224 ERROR_SMG_NO_TARGET_WINDOW
418 ERROR_SMG_INVALID_CALL
458 ERROR_SMG_INVALID_STOP_OPTION
459 ERROR_SMG_BAD_RESERVE
460 ERROR_SMG_PROCESS_NOT_PARENT
463 ERROR_SMG_RETRY_SUB_ALLOC

0 STOP_SESSION_SPECIFIED
1 STOP_SESSION_ALL

- ✓ Can only be used by a parent session to stop one or all of its child sessions.
- ✓ If the specified child session has related sessions, the related sessions are also terminated.
- ✓ The parent session may be running in the foreground or background.
- ✓ If the child session is running in the foreground when it is terminated, the parent session becomes the foreground session.

`DosStopSession (STOP_SESSION_ALL, idSession);`

20SESN12 940125

Pipes

Anonymous Pipes

Standard Input/Output

Duplicating File Handles

Named Pipes

20PIPE01 920810

Standard Input/Output

'C' Example of Using Standard Input/Output

```
/* lower.c -- converts uppercase characters to lower case */
#include <stdio.h>
#include <ctype.h>
main (int argc, char * argv[], char * envp[])
{
    char ch;
    while (( ch = getchar() ) != EOF ) // read STDIN
        putchar ( tolower ( ch ) ); // write STDOUT
}
```

- ① Standard devices can be redirected to other devices, files or pipes.
- ① A child process could inherit Standard device handles that have been redirected to pipe handles.

20PIPE03 930810

— unnamed pipes are half duplex

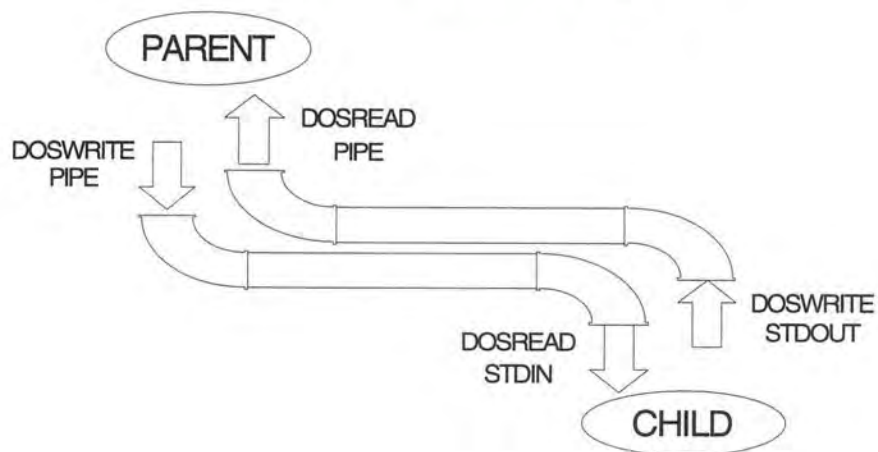
Anonymous Pipes

- ✓ A data storage buffer that OS/2 maintains.
- ✓ The maximum size of the buffer is 64 Kb.
- ✓ Similar to a file in that it is read and written via the file handle operations DosRead and DosWrite.
- ✓ Differs from a file in that the data may only be accessed in FIFO fashion.
- ✓ Since a child process inherits file handles from the parent, anonymous pipes are used to communicate between related processes.

20PIPE02 921101

NOT RANDOM ACCESS

Using Anonymous Pipes



- ✓ Two pipes must be created to provide bidirectional communication.
- ✓ Standard device handles are redirected to the pipe read/write handles with `DosDupHandle`.

DosCreatePipe, DosDupHandle

DosCreatePipe (PHFILE phfRead, --> ptr to returned read handle
PHFILE phfWrite, --> ptr to returned write handle
ULONG cb); <-- size of pipe buffer

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY

```
DosCreatePipe ( &hfRead, &hfWrite, 1024 );
```

DosDupHandle (HFILE hFile, <-- handle to be duplicated
PHFILE phFile); <--> ptr to duplicate

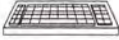


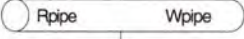






0	NO_ERROR
4	ERROR_TOO_MANY_OPEN_FILES
6	ERROR_INVALID_HANDLE
114	ERROR_INVALID_TARGET_HANDLE

```
HFILE hNewHandle = STDIN;
```

```
DosDupHandle ( hfRead, &hNewHandle );
```

20PIPE05 921021

Pipes/Redirection Example

STDIN(0)	STDOUT(1)	STDERR(2)	Handle 6	Handle 7	Handle 8	
			X	X	X	Start
"	"	"			X	CreatePipe
"	"	"	"	"		DupHandle (0, &-1)
	"	"	"	"	"	DupHandle (6, &0)
"	"	"	No Inherit	No Inherit	No Inherit	DosSetFHState ()
"	"	"	"	"	"	ExecPgm
	"	"	"	"	"	DupHandle (8, &0)
"	"	"	"	"	X	DosClose (8)
"	"	"	"	"	"	DosWrite (7) <u>parent</u>
						<u>DosRead (STDIN) child</u>

20PIPE06 921027

— Example of setting up a child's `STDIN(0)` as Parent's `STDOUT(1)`

Named Pipes

- ✓ Like anonymous pipes, they provide serial communication between two processes.
- ✓ Users of the named pipe need not be related.
- ✓ Named pipes can be used in byte-stream or message mode.
- ✓ Named pipes may be full duplex. A single handle is used for both reading and writing.
- ✓ Named pipes work both locally on a single system and remotely across a network.
- ✓ DOS workstations may communicate with an OS/2 server through named pipes.

20PIPE06 920810

Server/Client Communication Using Named Pipes

- ✓ The server process creates the pipe by calling `DosCreateNPipe`. The pipe is now 'disconnected' and cannot be opened by a client process.
- ✓ The server process calls `DosConnectNPipe` to put the pipe into the 'listening' mode.
- ✓ A client process supplies the name of the pipe in a call to `DosOpen` and receives a pipe handle. The pipe is now in the 'connected' state.
- ✓ The server and client process communicate by calling `DosRead`, `DosWrite` or `DosTransactNPipe`.
- ✓ The client process calls `DosClose` to close its end of the pipe. The pipe is now in the 'closing' state and cannot be accessed by another client.
- ✓ The server process calls `DosDisconnectNPipe`. The pipe is now in the 'disconnected' state again.
- ✓ To enable another client, the server must call `DosConnectNPipe` again.

20PIPE08 021124

Named Pipe Server Program Example

```
// example multi-thread server program for Named Pipes    pipeserv.c

#define INCL_BASE
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#define PIPE_NAME        "\\pipe\\example"
#define LOWER_CASE_OFFSET ('a' - 'A')
#define STDOUT          1

HEV    server_sem = 0;
USHORT connection_id = 0;
HPIPE  pipe_handle;
BOOL    flContinue = TRUE;

// internal functions
void thread (void);

// external functions
void terminate ( PSZ pszMsg, APIRET rc, PSZ pszMyName);

int main ( void )
{
    TID    thread_id;
    PSZ    pszMyName = "PIPESERV.";
    APIRET rc;

    while ( flContinue )
    {
// create an instance of the named pipe
        rc = DosCreateNPipe ( PIPE_NAME, &pipe_handle, NP_ACCESS_DUPLEX,
                               NP_WAIT | NP_READMODE_MESSAGE |
                               NP_TYPE_MESSAGE | NP_UNLIMITED_INSTANCES,
                               128, 128, 0 );
        if (rc) terminate ("Error Creating Pipe\n", rc, pszMyName);

        DosEnterCritSec ();
        printf ("%s Pipe Instance Created. Waiting for a \"\n",
                "connection...\n", pszMyName);
        DosExitCritSec ();

// put pipe in 'listening' mode and wait for a connection
        rc = DosConnectNPipe ( pipe_handle );
        if (rc) terminate ("Error Connecting Pipe\n", rc, pszMyName);
        printf ("%s Connection Received\n", pszMyName);

// create event sema4. Server thread will post when it starts
        rc = DosCreateEventSem ( 0, &server_sem, 0, FALSE );
        if (rc) terminate ("Error Creating EventSem\n", rc, pszMyName);

// create a server thread
        rc = DosCreateThread ( &thread_id, (PFNTHREAD) thread, 0, 0, 8192 );
        if (rc) terminate ("Error Creating Server Thread\n", rc, pszMyName);
        printf ("%s Server Thread Created\n", pszMyName);

// wait for signal that thread is running
        rc = DosWaitEventSem ( server_sem, 1000 );
        if (rc) terminate ("Error waiting for thread sema4\n", rc, pszMyName);
    } // end while

    return 0;
}
```

20PIPE09 921129

Named Pipe Server Program Example cont'd

```
// thread to manage a server connection          pipeserv.c
void thread ( void )
{
    APIRET rc;
    USHORT my_connection;
    HPIPE hPipe;
    BOOL keep_connection = TRUE;
    CHAR buffer[128];
    ULONG bytes_read;
    USHORT index = 0;
    ULONG bytes_written;
    PSZ pszMyName = "SERVTHRD:";

    // bump the connection count
    connection_id += 1;

    // get my connection id
    my_connection = connection_id;

    // get my pipe handle
    hPipe = pipe_handle;

    // signal the main thread so it can continue
    printf ("%s Connection #%d\n", pszMyName, my_connection);
    DosPostEventSem ( server_sem );

    while ( keep_connection )
    {
        // read a message from the pipe
        rc = DosRead ( hPipe, buffer, 128, &bytes_read );
        if (rc)
        {
            keep_connection = FALSE;
            break;
        }
        // print message
        DosEnterCritSec ();
        printf ("%s%d Message received: %s\n",
            pszMyName, my_connection, buffer );
        DosExitCritSec ();

        // process and write it back
        while ( index < strlen (buffer) )
        {
            if (( buffer[index] >= 'a' ) && ( buffer[index] <= 'z' ))
                buffer[index] -= LOWER_CASE_OFFSET;
            index += 1;
        } // end while

        rc = DosWrite (hPipe, buffer, strlen (buffer), &bytes_written );
        if (rc) terminate ("Error Writing Pipe\n", rc, pszMyName);

        // if 'done', drop thru and disconnect
        keep_connection = (strcmp (buffer, "done") != 0 );
    } // end-while

    // disconnect and close the pipe
    rc = DosDisconnectNPipe ( hPipe );
    if (rc) terminate ("Error Disconnecting Pipe\n", rc, pszMyName);
    printf ("%s%d Disconnecting\n", pszMyName, my_connection);

    // close the pipe
    rc = DosClose ( hPipe );
    if (rc) terminate ("Error Closing Pipe\n", rc, pszMyName);

    // exit the thread
    DosExit ( EXIT_THREAD, 0);
}
```

20PIPE10.921129

Named Pipe Client Program Example

```
// example client program for Named Pipes
// pipeclnt.c
#define INCL_BASE
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define STDOUT 1

// external functions
void terminate ( PSZ pszMsg, APIRET rc, PSZ pszMyName);

int main ( void )
{
    APIRET rc = 0;
    HPIPE pipe;
    ULONG action;
    CHAR pipe_name[61];
    CHAR input[64];
    PSZ pszMyName = "PIPECLNT:";
    CHAR result[64];
    ULONG bytes_returned;

    // get name server. If no server name entered, connect locally
    printf ("Enter the server name or <Enter>: \n");
    gets (input);
    if (input[0] != '\0' )
    {
        strcpy (pipe_name, "\\");
        strcat (pipe_name, input);
        strcat (pipe_name, "\\pipe\\example");
    }
    else
        strcpy(pipe_name, "\\pipe\\example");

    // wait for available instance of pipe
    rc = DosWaitNPipe ( pipe_name, SEM INDEFINITE WAIT );
    if (rc) terminate ("Error waiting for pipe instance\n", rc, pszMyName);
    printf ("%s Pipe Available\n", pszMyName);

    rc = DosOpen ( pipe_name, &pipe, &action, 0, 0,
        OPEN_ACTION_OPEN_IF_EXISTS,
        OPEN_ACCESS_READWRITE | OPEN_SHARE_DENYREADWRITE |
        OPEN_FLAGS_WRITE_THROUGH, 0 );
    if (rc) terminate ("Error opening pipe\n", rc, pszMyName);
    printf ("%s Pipe Opened\n", pszMyName);

    rc = DosSetNPHState ( pipe, NP_READMODE_MESSAGE ); // chg. mode
    if (rc) terminate ("Error changing Pipe Mode\n", rc, pszMyName);
    printf ("%s Pipe Mode Changed\n", pszMyName);

    while ( strcmp ( input, "done" ) != 0 )
    {
        printf ("Enter string to transmit:\n");
        gets ( input );

        rc = DosTransactNPipe ( pipe, input, strlen (input) +1, result,
            64, &bytes_returned );
        if (rc) terminate ("Error in TransactNPipe\n", rc, pszMyName);
        result[bytes_returned] = '\0';
        printf ("Result: %s\n", result );
    }
    rc = DosClose ( pipe );
    if (rc) terminate ("Error Closing Pipe\n", rc, pszMyName);
    printf ("%s Closing the Pipe\n", pszMyName);
    return 0;
}
20PIPE11 921208
```

Summary of Named Pipe Functions

DosCallNPipe
DosConnectNPipe
DosCreateNPipe
DosDisConnectNPipe
DosPeekNPipe
DosQueryNPHState

DosQueryNPipeInfo
DosQueryNPipeSemState
DosSetNPHState
DosSetNPipeSem
DosTransactNPipe
DosWaitNPipe

- _beginthread 9-21, 9-22
- _CRT_init 16-23, 16-25
- _DLL_InitTerm 16-23
- _endthread 9-21
- DosAddMuxWaitSem 10-20
- DosAllocMem 7-6, 7-10
- DosAllocSharedMem 13-2, -3, -4, -5, 15-2
- DosAsyncTimer 11-4
- DosBeep 4-4, 4-6, 4-7, 4-8
- DosClose 14-14, 19-6, 19-8
- DosCloseEventSem 10-7, 14-10
- DosCloseMutexSem 10-13
- DosCloseMuxWaitSem 10-19
- DosCloseQueue 15-2, 15-8
- DosConnectNPipe 19-8
- DosCreateDir 14-24
- DosCreateEventSem 10-5, 15-9
- DosCreateMutexSem 10-11
- DosCreateMuxWaitSem 10-17, 15-9
- DosCreateNPipe 19-8
- DosCreatePipe 19-5, 19-6
- DosCreateQueue 15-2, 15-3, 15-9
- DosCreateThread 9-9, 9-11, 9-22, 14-10
- DosDeleteMuxWaitSem 10-20
- DosDisconnectNPipe 19-8
- DosDupHandle 19-4, 19-5, 19-6
- DosEnterCritSec 9-9, 9-12
- DosEnterMustComplete 17-7
- DosEnumAttribute 14-24
- DosError 17-6
- DosExecPgm 12-5, 19-6
- DosExit 4-5, 4-6, 9-9
- DosExitCritSec 9-9, 9-12
- DosExitList 12-15, 12-16
- DosExitMustComplete 17-7
- DosFindFirst 14-24
- DosFindNext 14-24
- DosFreeMem 7-7, 7-10, 13-3, 13-4, 15-2
- DosFreeModule 16-17, 16-19
- DosGetInfoBlocks 5-2, 9-9, 9-10
- DosGetNamedSharedMem 13-2, 13-3, 13-6
- DosGetSharedMem 13-2, 13-4, 13-8
- DosGiveSharedMem 13-2, 13-4, 13-7, 15-2
- DosKillProcess 12-9, 17-3
- DosKillThread 9-9, 9-14
- DosLoadModule 16-17, 16-18
- DosOpen 14-8, 14-24, 19-8
- DosOpenEventSem 10-6
- DosOpenMutexSem 10-12
- DosOpenMuxWaitSem 10-18
- DosOpenQueue 15-2, 15-4
- DosPeekNPipe 19-12
- DosPeekQueue 15-6
- DosPostEventSem 10-8, 14-10
- DosPurgeQue 15-8
- DosQueryEventSem 10-10
- DosQueryFHState
- DosQueryFileInfo 14-15, 14-24
- DosQueryMem 7-8
- DosQueryMutexSem 10-15
- DosQueryMuxWaitSem 10-22
- DosQueryNPHState 19-12
- DosQueryNPipeInfo 19-12
- DosQueryNPipeSemState 19-12
- DosQueryPathInfo 14-15, 14-24
- DosQueryProcAddr 16-17, 16-19
- DosQueryQueue 15-8
- DosRaiseException 17-5
- DosRead 14-9, 14-10, 19-6, 19-8
- DosReadQueue 15-2, 15-7, 15-9
- DosReleaseMutexSem 10-14
- DosRequestMutexSem 10-14
- DosResetBuffer 14-14
- DosResetEventSem 10-8
- DosResumeThread 9-9, 9-13
- DosScanEnv 5-3, 5-7
- DosSelectSession 18-11
- DosSendSignalException 17-4
- DosSetExceptionHandler 17-7
- DosSetFHState1 9-6
- DosSetFileInfo 14-24
- DosSetFileLocks 14-13
- DosSetFileSize 14-11
- DosSetMem 7-9
- DosSetPathInfo 14-24
- DosSetPriority 9-9, 9-15, 12-9
- DosSetSession 18-9, 18-10
- DosSetSignalExceptionFocus 17-4
- DosSleep 11-1
- DosStartSession 18-7, 18-8
- DosStartTimer 11-2
- DosStopSession 18-12
- DosStopTimer 11-5
- DosSubAllocMem 7-10, 7-12
- DosSubFreeMem 7-10, 7-12
- DosSubSetMem 7-10, 7-11
- DosSubUnsetMem 7-10, 7-13
- DosSuspendThread 9-9, 9-13
- DosTransactNPipe 19-8, 19-12
- DosUnsetExceptionHandler
- DosWaitChild 12-9, 12-10
- DosWaitEventSem 10-9, 14-10
- DosWaitMuxWaitSem 10-21, 15-9
- DosWaitThread 9-9, 9-15
- DosWrite 4-9, 14-9, 14-10, 19-6, 19-8
- DosWriteQueue 15-2, 15-5

- C Programming the OS/2 2.0 Environment, V. Mitra Gopaul, Van Nostrand Reinhold
Client-Server Programming with OS/2 2.0, Orfali & Harkey, G325-0650-01
Comprehensive Database Performance for OS/2's Extended Services, Bruce Tate, Tim Malkemus and
Terry Gray, G362-0012
IBM C Set/2 and IBM WorkFrame/2, S10G-4449
IBM C Set/2 Debugger Tutorial, S10G-4447
IBM C Set/2 Migration Guide, S10G-4445
IBM C Set/2 Reference Summary, S10G-4446
IBM C Set/2 User's Guide, S10G-4444
IBM Developer's Toolkit for OS/2 2.0 - Getting Started, 10G6199
IBM OS/2 2.0 Application Design Guide, S10G-6260
IBM OS/2 2.0 Control Program Programming Reference, S10G-6263
IBM OS/2 2.0 CUA Guide to User Interface Design, SC34-4289-0
IBM OS/2 2.0 CUA Interface Design Reference, SC34-4290-0
IBM OS/2 2.0 Information Presentation Facility Guide and Reference, S10G-6262
IBM OS/2 2.0 Physical Device Driver Reference, S10G-6266
IBM OS/2 2.0 PM Programming Ref. Vol. I, S10G-6264
IBM OS/2 2.0 PM Programming Ref. Vol. II, S10G-6265
IBM OS/2 2.0 PM Programming Ref. Vol. III, S10G-6272
IBM OS/2 2.0 Presentation Driver Reference, S10G-6267
IBM OS/2 2.0 Procedures Language 2/REXX Reference, S10G-6268
IBM OS/2 2.0 Procedures Language 2/REXX User's Guide, S10G-6269
IBM OS/2 2.0 Programming Guide Vol. I, S10G-6261
IBM OS/2 2.0 Programming Guide Vol. II, S10G-6494
IBM OS/2 2.0 Programming Guide Vol. III, S10G-6495
IBM OS/2 2.0 Red Book - Applications Development, GG24-3774
IBM OS/2 2.0 Red Book - Control Program, GG24-3730
IBM OS/2 2.0 Red Book - DOS & Windows Environment, GG24-3731
IBM OS/2 2.0 Red Book - Presentation Manager & Workplace Shell, GG24-3732
IBM OS/2 2.0 Red Book - Print Subsystems, GG24-3775
IBM OS/2 2.0 System Object Module Guide and Reference, S10G-6309
IBM OS/2 2.0 Virtual Device Driver Reference, S10G-6310
Inside OS/2 Release 2.0, Staff of New Rider Publishing, G362-0016-00
Integrating Applications with OS/2.0, William H. Zack
Learn to Program OS/2 Presentation Manager by Example, Stephen A. Knight, G362-0011
Now That I Have OS/2 2.0 on My Computer -- What Do I Do Next?, Steven Levenson, G362-0008-00
OS/2 Presentation Manager GPI: A Programming Guide to Text, Graphics and Printing, Graham
Winn, G362-0005
OS/2 Update, Dick Conklin, Microsoft Press
SAA Common Programming Interface, C Reference Level 1, SC09-1308
The COBOL Presentation Manager Programming Guide, David M. Dill, G362-0010
The Design of OS/ 2 - 32-Bit, Dietel & Kogan, Addison Wesley, S325-4005-00
The OS/2 2.0 User' Guide for the Workplace Shell, Maria Tyne, G362-0014
Using OS/2 2.0, Barry Nance & Greg Chicares, G362-0007
Writing OS/2 2.0 Device Drivers in C, Steven J. Mastrianni, G362-0006