

Windows -- NT (New Technology)

- Multi-threading
- Pre-emptive multi-tasking, 0-15 dynamic, 16-31 real time
- Demand paged VM (2G user space, server 3G user)
- Global cache
- multi-processor
- processor independent (HAL)
- Microkernel based
- Integrate Networking
- Compatible with older Windows
- Multiple OS emulation
- Performance
- code separation: executive, OS sub systems ...
 - Env: POSIX, OS/2, DOS, WIN32,
 - 4.0: WIN32 into kernel, POSIX, OS/2 reduced
 - most UNIX emulation now done via WIN32 (cygwin)

Windows (page 2)

Windows 2000 -- NT 5.0

- Added WDM -- Windows Driver Model
- A lot more plug-and-play

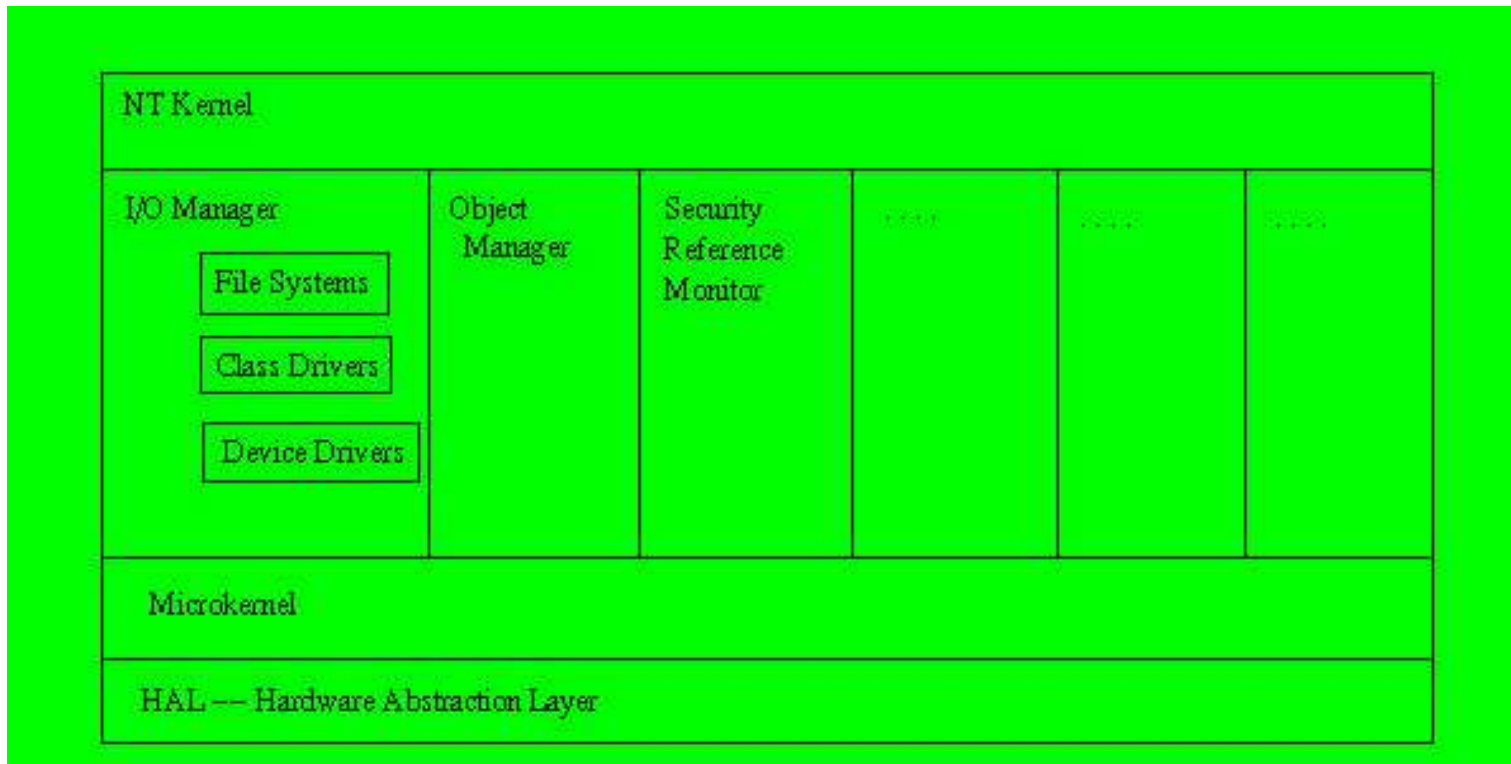
Windows XP -- NT 5.1

- A lot of power management
- In the kernel, minor upgrade
- Reason our book is written for W2000

Windows Vista -- NT 6.0

- Ignore it for this class

Kernel design



- HAL - Hardware Abstraction Layer, Microkernel
- NT is object oriented, C++, ...
- Collection of Managers
 - I/O Manager
 - Object Manager
 - Security Manager
 - Cache Manager
 - And so forth ...

NT Kernel & Driver Model



- I/O Manager Driver structure
 - File Systems
 - Class Drivers (e.g. disk, functional)
 - Device Drivers (e.g. ide disk, physical)
 - Mini Drivers (various)

I/O Manager

- All drivers dynamically loaded

- I/O requests -> I/O manager
 - IRP: I/O Request Packet
 - Process IRPs
 - May queue an IRP
 - "Execution Context" - not always user

Object Manager

- name space for all objects
- keeps track of all objects, some not named
- Internal name space different than normal user view

Example names:

```
// win 32 version  
hfile = CreateFile("C:\\foo\\bar");
```

```
// System call version  
... NtCreateFile ( ... "\\??\\C:\\foo\\bar" ...)
```

I/O manager -> Object Manager
then to driver "stack"

kernel routines may call: e.g. ZwCreateFile (...)

HAL

Provides abstractions for:

- device addressing
- I/O architecture
- Interrupt management
- DMA management
- DMA operations
- System clocks and Timers
- Firmware & BIOS interfacing
- Configuration Management
- HAL changes

Device Addressing

- HAL provides a logical address
- READ_PORT_{UCHAR,USHORT,ULONG} (address)
- READ_REGISTER_{UCHAR,USHORT,ULONG} (address)
 - address is: base + offset
- WRITE_PORT_{UCHAR,USHORT,ULONG} (address, value)
- WRITE_REGISTER_{UCHAR,USHORT,ULONG} (address, value)
 - address is: base + offset
- No reference to bus is used.
- Must be directly connected bus like ISA, PCI, PCMCIA
- Does provide HalGetBusData, HalSetBusData, HalAssignSlotResources

Registry ...

- Key / data pair database
 - keys are hierarchical
 - may have sub keys
- uses wide character strings
- unicode string: struct with: length, maxlength, buffer

- Device Driver interest ...
- Device needs an entry in registry
- Driver needs an entry in registry
- HKEY_LOCAL_MACHINE\SYSTEM
 - control sets ...
- Specific keys and entries later ...

Kernel-Mode I/O Processing (Chapter 3)

- Execution Context
 - trap or exception
 - interrupt
 - kernel thread

- Interrupt levels
 - HIGHEST_LEVEL
 - POWER_LEVEL, IPI_LEVEL, CLOCK{2,1}_LEVEL, PROFILE_LEVEL
 - DIRQLs -- device interrupt levels
 - DISPATCH_LEVEL -- thread schedule, DPCs
 - APC_LEVEL -- APC
 - PASSIVE_LEVEL -- Lowest level

Interrupt processing in NT

- Interrupt at a DIRQL
- Do minimal work here
- Schedule a Deferred Procedure Call (DPC) to finish work
 - Max of 1 DPC / driver in queue

- Access to user buffers
 - buffered i/o -- driver uses system buffers
 - direct i/o -- user pages wired down, use kernel pointers
 - no uiomove equivalent (to my knowledge!)

Outline of a Kernel-Mode Driver

- Entry points
 - Driver Entry
 - Reinitialize
 - Unload
 - Shutdown
 - Bugcheck
- I/O system service Dispatch routines
 - request function code / dispatch table
 - Open, Close -- required
 - Read, Write, DeviceIoControl -- optional
- More specific details later
- Preemptible
- Interruptible
- Reentrant

Data Transfer ...

Typical I/O style .. e.g. floppy

- Read/Write IRP, queue IRP (like tsleep)
- StartIO routine ... to start actual I/O
- ISR -- when I/O done
- DPC -- post ISR processing
- DPC may trigger next StartIO

Resource Sync -- routines to help with

- Reentrant code
- multi-thread, multi-cpu
- controller, multi-device methods
- DMA adapter ...
- interrupt protection ...
- actual routines later ...

Others -- timer, I/O completion, I/O Cancel ...

I/O processing sequence

- preprocessing by the I/O manager
- preprocessing by the device driver
- device start, interrupt service
- postprocessing by the driver
- postprocessing by the I/O manager

Read the book for details ...

Drivers & Kernel-mode objects (CH 4)

Pseudo objects ... not really classes, just structs

BSD uses struct in struct ... e.g. softc

Windows uses void pointers => lots of structs

Kernel uses pointers to "objects" in many places

I/O Request Packet (IRP)

- Header & Stack
- Header -- general bookkeeping
 - IoStatus
 - System buffer pointer (Buffered I/O)
 - Memory Descriptor List (Direct I/O)
 - Cancel

I/O Request Packet (page 2)

- Stack -- per driver information
 - Major function
 - Minor function
 - Union based on function ...
 - read struct (length, key, byteoffset)
 - write struct (length, key, byteoffset)
 - deviceioctl struct (....)
 - others ... PVOID (void *)
 - Pointer to Device Object
 - Pointer to File Object (if any)
 - Pointer to completion routine
- One stack entry for each driver to process IRP
- Kernel has set of routines that work on IRPs
 - IoStartPacket, IoCompleteRequest, IoStartNextPacket,
 - IoCallDriver, IoAllocateIrp, IoFreeIrp, ...

Driver Objects

- I/O Manager creates
- Passed to DriverEntry

- Contents ...
 - DeviceObject list
 - Pointers to routines
 - StartIO, AddDevice, Unload, ...
 - MajorFunction dispatch table

- DriverEntry initializes
 - "Kernel-mode" ... builds device list here
 - "WDM" uses AddDevice to build device list

Device Object

- 1 per virtual, logical and physical device
- Created by DriverEntry or AddDevice
- Contents
 - Next Device pointer
 - Driver Object pointer
 - Flags
 - CurrentIRP pointer
 - Device Extension pointer
 - like the softc (minus the device struct)
 - pending IRP queue

Device Extension

- Per device driver specific information
- contents ?
 - pointer to device object
 - any thing else needed ... buffer pointers, ...
- declared by driver

Other Objects

- Controller Object
 - multiple devices, one controller, e.g. disks
 - pointed to by device extension
 - serializes access to controller
- Controller Extensions
 - Controller specific information
- Adapter Objects
 - Typically used for DMA access
 - serializes access to DMA hardware
- Interrupt Objects
 - Created by DriverEntry or AddDevice
 - Interrupt Object has a pointer to ISR code
 - Pointer to Interrupt Object in device or controller extension

General Development Issues (CH 5)

- Similar ideas to development in NetBSD ...
- Environment
 - DDKs, SDKs
 - Visual Studio vs. Build environments
 - Free vs Checked
- Microsoft Names ... NTDDK.h, WDM.h
 - PVOID => void *
 - WCHAR => wchar_t
 - PWSTR => wchar_t*
 - Many others
- Sample drivers
 - DDK
 - Book's CD

General Development Issues (page 2)

- Status Return Values
 - kernel routines return NTSTATUS
 - STATUS_SUCCESS, STATUS_...
 - (Not win32 ERROR_XXX)
- Supporting routines ... see DDK documentation
 - Nice table on p85
 - IoXXX -> I/O manager
 - KeXXX -> kernel services, typically sync
 - MmXXX -> memory manager
 - RtlXXX -> general runtime library (e.g. no C lib)
 - ZwXXX -> misc, typically OS access to sys calls
- Tweaks for commercial code
 - making "init" code unloadable
 - allowing driver to be paged

Kernel Memory

- Kernel Stack
 - Limited to 16KB, no buffers, no deep recursion ...
- Paged Area (Pool)
 - Access only at DISPATCH_LEVEL and lower
 - May cause a page fault
 - No access by interrupt routines ...
- Nonpaged Area (Pool)
 - Access at all IRQLs
 - device/controller extensions here
- Variety of routines to access
 - NonPagedPool(), NonPagedPoolMustSucceed(),
 - NonPagedPoolCacheAligned(), PagedPool()...
- Typical size is PAGE_SIZE bytes

More Memory

- Smaller hunks ...
 - Zone Buffers
 - Allocate large chunk
 - Routines break up large chunk into smaller sizes
 - NT provides a variety of routines to manipulate

- Programmer could write own routines for
 - malloc / free
 - new / delete
- Small hunks inside larger kernel hunks

Unicode Strings

- Kernel uses Unicode strings
- UNICODE_STRING, PUNICODE_STRING
 - struct { USHORT len, USHORT maxlen, PWSTR buffer }
- C: L"some text"
- Collection of RtlXXX functions for unicode (p93)
 - RtlInitUnicodeString(), RtlAnsiStringToUnicodeString(), ...

Interrupt blocking

- Windows also does interrupt locking ...
 - KeRaiseIrql(), KeLowerIrql(), KeGetCurrentIrql()

DPC synchronization

- Allow DPC to do share data structure access
- kernel does serialization, multi-cpu sync also

Multiple CPU

Spin Lock

- Interrupt Spin Locks
 - e.g. DIRQL on one CPU doesn't stop other CPUs.
 - Be careful about multi-cpu entry in routines
 - Spin Lock causes other CPUs to spin ...
- Executive Spin Locks
 - Access to OS data structures with multi-cpu access
- Routines:
 - KeInitializeSpinLock(), KeAcquireSpinLock(),
 - KeReleaseSpinLock(), ...
 - Raise IRQL to DISPATCH LEVEL
 - Typical "don't hold long"

Kernel has support for singly linked lists and doubly linked lists

Initialization and Cleanup Routines (Ch 6)

Start simple ... add stuff

NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDriverObject,
IN PUNICODE_STRING pRegistryPath)

□ General Steps

- Initialize Driver Object (pointers, dispatch table)
- If a controller object needed, create it.
- non WDM:
 - locate and claim hardware
 - Create Device object (IoCreateDevice())
 - IoCreateSymbolicLink() to win32 namespace
 - repeat last 2 for each device
- return STATUS_SUCCESS (if no errors)

DriverEntry

- `pDriverObject->DriverStartIo = toyStartIO;`
- `pDriverObject->DriverUnload = toyUnload;`
- `pDriverObject->DriverAddDevice = dasAddDevice;`
- `pDriverObject->MajorFunction [IRP_MJ_CREATE] = toyCreate;`
- ... for each `IRP_MJ_XXXX` the driver supports
- (more details in Ch 7)

Creating Device Objects

IoCreateDevice (IN PDRIVER_OBJECT pDrvObj,
IN ULONG DevExtSize, IN PUNICODE_STRING pDevName,
IN DEVICE_TYPE DevType, IN ULONG DevFlags,
IN BOOLEAN Exclusive, OUT PDEVICE_OBJECT *pDevObj)

□ Names

- \Device\devicenameN

- \?? win32 -- symbolic link here

 - IoCreateSymbolicLink

- winobj -- www.sysinternals.com (redirect)

□ DeviceType (see ntddk.h)

- FILE_DEVICE_UNKNOWN

- FILE_DEVICE_DISK

- 0x0000->0x7FFF MS, 0x8000->0xFFFF customer

□ DevFlags

- FILE_READ_ONLY_DEVICE ...

Minimal Driver Example (Non-WDM, Chap6)

Other Driver Routines

- Reinitialize
 - Wait for later
 - IoRegisterDriverReinitialization()
 - Reinitialize may use above
 - Called at PASSIVE_LEVEL
- Unload
 - delete all objects ...
 - turn off interrupts
 - deletes symbolic links
 - free any memory
- Shutdown
 - Doesn't need to free stuff
 - "stop device"

Hand installing drivers (e.g. toy ...)

\WINDOWS\System32\Drivers\toy.sys

Registry

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services

□ 3 DWORD values for ... \toy

□ ErrorControl

□ 0 => ignore, 1 => normal, 2 => severe, 3 => critical

□ Start

□ 0 => boot, 1 => system

□ 2 => auto, 3 => manual, 4 => disabled

□ Type

□ 1 => kernel mode driver, 2 => file system driver

□ 4 => network adapter, 8 => file system recognizer

Starting the driver

□ computer management console

□ net command

Dispatch Routines (Ch 7)

Major Function dispatch table

- IRP_MJ_XXXX
 - CREATE, CLEANUP, CLOSE, READ, WRITE, DEVICE_IOCTLCONTROL, INTERNAL_DEVICE_CONTROL, QUERY_INFORMATION, SET_INFORMATION, FLUSH_BUFFERS, SHUTDOWN, ...
- point at one routine or many
- pDO->MajorFunction [IRP_MJ_CREATE] = ToyCreate; ...
- Don't support it? Don't assign anything!
- Required?
 - CREATE, CLOSE
 - Anything by lower driver
- Prototype:
 - NTSTATUS name (IN PDEVICE_OBJECT pDO, IN PIRP pIrp)

Dispatch Routines (page 2)

- IRP is shared ... serially with I/O manager
- All other data structures (Do, Dext) need sync
 - KeSynchronizeExecution()
- General duties
 - IoGetCurrentIrpStackLocation() -- current stack entry
 - Parameter validation
 - intermediate level: may need to split IRP for lower level
 - Process the IRP ... as much as possible.
- Exit properly
 - IoStatus.Status field in IRP (set)
 - Error => clear IoStatus.Information fields
 - IoCompleteRequest() (often no priority increment)
 - return same status as in IRP
 - Not: return pIrp->IoStatus.Status
 - Irp deallocated by IoCompleteRequest!

Code fragments:

```
NTSTATUS DispatchWrite (IN PDEVICE_OBJECT pDO,  
                      IN PIRP pIrp) {  
    pIrp->IoStatus.Status = STATUS_NOT_SUPPORTED;  
    pIrp->IoStatus.Information = 0;  
    IoCompleteRequest (pIrp, IO_NO_INCREMENT);  
    return STATUS_NOT_SUPPORTED;  
}
```

```
NTSTATUS DispatchClose (IN PDEVICE_OBJECT pDO,  
                      IN PIRP pIrp) {  
    .... other code ....  
    pIrp->IoStatus.Status = STATUS_SUCCESS;  
    pIrp->IoStatus.Information = 0;  
    IoCompleteRequest (pIrp, IO_NO_INCREMENT);  
    return STATUS_SUCCESS;  
}
```

Scheduling a Device Operation

- If can't immediately do operation ... schedule it
- IoMarkIrpPending
- IoStartPacket (send it to StartIO routine)
- Return STATUS_PENDING
- Note: Must call one of following before return
 - IoMarkIrpPending
 - IoCompleteRequest

Reads & Writes ...

- Buffering: stated in Flags field of Irp
 - Buffered I/O -- system buffer
 - AssociatedIrp.SystemBuffer
 - Ignore UserBuffer field ...
 - Direct I/O -- addresses to pinned user pages
 - MdlAddress
 - MMGetSystemAddressForMdl() -- programmed I/O

Reads & Writes (more)

- Buffering: more
 - Neither I/O -- only user space addresses
 - Not very useful ... only for dispatch of highest level driver
- Transfers:
 - May be done directly by dispatch routines
 - May be done by StartI/O
 - May be done by DpcForIsr ...
 - Possibly done by Isr ...
 - May be done by DMA!

Loopback Example! Chapter 7, Book's CD.

DeviceIoControl

- Weird operations ... e.g. set channel, set rate ...
- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_INTERNAL_DEVICE_CONTROL
- Irp->Parameters.DeviceIoControl.IoControlCode
 - encodes device type, transfer type, access and code
- IOCTL value (same as IoControlCode field)
- Device Type (bits 31-16)
 - Same as to IoCreateDevice
- Required Access (bits 15, 14)
 - FILE_ANY_ACCESS
 - FILE_{READ,WRITE}_DATA (read | write)
- ControlCode (bits 13-2)
 - 0x000 - 0x7ff MS, 0x800 - 0xfff customer
- TransferType (bits 1, 0)
 - METHOD_BUFFERED, METHOD_NEITHER
 - METHOD_IN_DIRECT, METHOD_OUT_DIRECT

IOCTL Header files

- Code definitions
 - #include <WINIOCTL.h>
 - #define IOCTL_MY_CTL CTL_CODE(Type, code, transfer, rwaccess)
- Use code definitions in both user code and driver code
- ... switch (ControlCode) ... case IOCTL_MY_CTL
- May process directly
- May mark pending and do a IoStartPacket()!
- IOCTL buffer management
 - BUFFERED ... r/w in one buffer
 - IN_DIRECT ... input via MDL, output via buffer
 - OUT_DIRECT ... input via buffer, output via MDL

Testing drivers to this point ... see book p139, p140

"Build a driver framework that is proven before adding hardware interaction." (e.g. A Toy!)

Interrupt Driven I/O (Ch 8)

- Programmed I/O vs DMA
- Both may require interrupts
- Programmed generic outline
 - IRP (write, read) -> queue
 - StartIO: init, start first op
 - ISR: may start next transfer
 - repeat as needed
 - ISR done, request DPC
 - DPC: marks IRP as done, starts next if needed
- Sync between above code paths ...
 - KeSynchronizeExecution -- calls a routine at DIRQL of INTR.
 - Critical section appears in a function.
 - bool KeSynchronizeExecution (pIntrObj, pRoutine, pVoid)
 - calls pRoutine(pVoid) at DIRPL of IntrObj
 - return value is same as pRoutine returns.

Simple DPC processing

- Simple version
 - IoInitializeDpcRequest (DriverEntry or AddDevice)
 - ISR: IoRequestDpc
- More complex
 - ISR may have to start one of several DPCs
 - more details later!
- IoCompleteRequest ... priority boost
 - IO_NO_INCREMENT, IO_DISK_INCREMENT, IO_SERIAL_INCREMENT, ...
IO_SOUND_INCREMENT
- Example ... parallel port driver (Chap8/PPort)
 - ClaimHardware()
 - HalGetInterruptVector()
 - IoConnectInterrupt()
 - IoDisconnectInterrupt()

Hardware Initialization ...

Plug and Play History

- ISA bus ... dumb bus controller
- PCI, PCMCIA, smarter bus controller
 - detect devices
 - assign resources (port space, irqs, ...)
- PnP designed to abstract smarter buses
 - Automatic detection of installed (and removed) devices
 - Drivers for (new) hardware automatically loaded by system
 - Deal with hot plugging of devices
 - "Software control" of configuration of devices

Components of PnP

- PnP Manager -- kernel & user mode parts
- Power Manager
- Registry
- INF Files
- PnP Drivers, WDM & NT ...

Plug and Play (Page 2)

- Legacy Drivers ..
 - probe for devices
 - run drivers at startup ...
 - "driver-centric"
- PnP
 - OS finds devices, starts associated driver
 - Driver may find more devices and start more drivers
 - "device-centric"
- Changes to Driver
 - DriverEntry just fills in pointers to routines.
 - New function ... AddDevice
 - IRP_MJ_PNP ... for all PnP related IRPs.
 - IRP_MN_START_DEVICE, ...
 - "Bus drivers" auto load other drivers, ...

Device Stack

WDM -- layers

- Physical Device Objects (PDOs)
- Functional Device Objects (FDOs)
- Filter Device Object

PDOs come in 2 flavors

- bus PDO
- nonbus PDO

Windows PnP startup

- 1. Find all buses (create bus PDOs)
- 2. Load drivers for all buses (create bus FDOs)
- 3. Bus drivers create PDOs for each connected device
- 4. Load drivers for PDOs (create nonbus FDOs / AddDevice)

Device Stack (page 2)

Routines used (usually in AddDevice)

- IoCreateDevice
- IoAttachDeviceToDeviceStack (at the top)

No hardware initialization done in AddDevice!
Just create objects! (and symbolic link them)

Actual Hardware Initialization comes in a different Pnp IRP

- IRP_MN_START_DEVICE
- IRP_MN_QUERY_STOP_DEVICE
- IRP_MN_CANCEL_STOP_DEVICE
- IRP_MN_STOP_DEVICE
- IRP_MN_QUERY_REMOVE_DEVICE, ...
- IRP_MN_SUPRISE_REMOVAL

Need to pass IRPs down the stack ...

Passing IRPs and Completion Routines

- IoCopyCurrentStackLocationToNext()
- IoCallDriver()
- IoSkipCurrentIrpStackLocation()

Example code:

```
IoCopyCurrentStackLocationToNext (pIrp);  
IoCallDriver (pDO->DeviceExt->PLowerDevice, pIrp);  
...
```

Or

```
IoSkipCurrentStackLocation (pIrp);  
return IoCallDriver (pDO->DeviceExt->PLowerDevice, pIrp);
```

Completion routines

- IoSetCompletionRoutine()
- IoCompleteRequest()

IoCopyCurrentStackLocationToNext(pIrp);

IoSetCompletionRoutine (pIrp, Func, ..)

Return IoCallDriver (...lowerdevice, pIrp)

Func(pDO, pIrp, pVoidContext) return pIrp->IoStatus.Status...

- PnP IRP execute at PASSIVE_LEVEL
- Hard to predict the IRQL of completion routine
- DISPATCH_LEVEL and above are restricted ... no page faults

Events ...

Another method of sync ... and to guarantee PASSIVE_LEVEL

```
KEVENT event;  
IoCopyCurrentStackLocationToNext(pIrp);  
KeInitializeEvent (&event, NotificationEvent, FALSE);  
IoSetCompletionRoutine (... func, &event ...)  
IoCallDriver ( ... lower device, pIrp)  
KeWaitForSingleObject (&event, Executive, KernelMode,  
    FALSE, NULL);  
... processing  
return status ....
```

```
Func:  
PEVENT pEvent = (PEVENT) pContext;  
KeSetEvent (pEvent, 0, FALSE);  
return STATUS_MORE_PROCESSING_REQUIRED;
```

Hardware Resource Descriptors

- Information Passed to IRP_MN_START_DEVICE
- Parameters.StartDevice.AllocatedResources
- Parameters.StartDevice.AllocatedResourcesTranslated
- Structure with lots of unions
- Count -- number of elements which follow
- ResourceList
 - FullDescriptor
 - PartialList
 - PartialDescriptor
- PartialDescriptor has:
 - Type
 - Unions for interrupt, dma, port, Memory
 - Interrupt: u.Interrupt.{Level,Vector,Affinity}
 - Port: u.Port.{Start,Length}

Processing IRP_MN_START_DEVICE

Get hardware information, store in device extension

For DAS ...

- (assume) one interrupt entry, two port entries
- Use the Port.Start as "base address" + offset
- No mapping required, just use the translated list!

Device Interfaces ...

This is a way to export a new API to userland to replace the DeviceIoControl method.

Review Chap 9 - Plug and Play sample driver

Full WDM in Chap 10 ... Power Management

System Threads (Chap 14)

Driver can create a system thread to run in the driver.

- PsCreateSystemThread()
- PsTerminateSystemThread()
- KeSetPriorityThread()
- System Worker Threads
 - ExInitializeWorkItem()
 - ExQueueWorkItem()
- KeDelayExecutionThread()

Event Objects

- KeInitializeEvent()
- KeWaitForSingleObject()
- KeWaitForMultipleObjects()
- KeSetEvent()
- KeResetEvent()
- KeClearEvent()
- Various others ...
 - shared events between drivers
 - mutex objects
 - semaphore objects
 - timer objects
 - thread objects

Driver Installation (Chap 16)

INF text files

Driver installation does 2 things:

- system registry entries ...
- Driver files copied

INF file structure

- [section]
- entry = value , value ...

Sections

- [Version] -- first
- [Manufacturers] -- required
 - manufacturer=model

More on INF files

- [model] -- for each model
 - device-description=install-section-name,
hw-id,compatible-id... (compat opt)
 - device-description human readable ...
- [Version]
 - Signature (\$WindowsNT\$, \$Windows 95\$, \$Chicago\$)
 - Class (opt) (predefined names see ddk doc)
 - ClassGuid (opt)
 - Provider (opt) organization name
 - LayoutFile (opt)
 - CatalogFile (opt) validated drivers ...
 - DriverVer mm/dd/yyyy[,x,y,v,z]

[Manufacturers]

manufacturer=model

- [model]

- device-description=install-section-name,hw-id
 ,compatible-id... (compat opt)

- [install-section-name] (aka [DDInstall])

- DriverVer

- CopyFiles=file-list-section, ... or @filename

- AddReg (req?)

- Include

- Needs (sections from include files)

- DelFiles (for upgrade deletes...)

- RenFiles (rename before install)

- DelReg (delete registry entries)

- ProfileItems (section names that modify start menu ...)

More sections

- [install-section-name.Services]
 - AddService= servicename, flags, service-install-section
- [CopyFiles] section
 - destination-filename [,source-filename, temp-filename, flag]
 - flags -> COPYFLG_NO_OVERWRITE, ...
- [SourceDisksNames]
- [SourceDiskFiles]
- [DestinationDirs]
 - DefaultDestDir=12 ; %windir%\system32\drivers

[AddReg] section

- reg-root[,subkey,value-name,flags,value]
 - HKCR, HKCU, HKLM, HKU, HKR
- Read the Book for all the details ...

Class Names and Device IDs (p363)

- <enumerator>\<enumerator-specific-device-id>
- PCI\VEN_1307&DEV_0029
 - &SUBSYS_00291307&REV_02
 - Possibly different for new cards
- Installer looks in [Models] for hw-id matches
- match implies use this .INF

Setup Classes, GUIDs ..

- "Ports" works
 - nothing else looks close
 - appears "uncool" to define your own
- Known to work:
 - Class = Ports
 - ClassGuid = {4d36e978-e325-11ce-bfc1-08002be10318}
- Simple example on page 359

