6. Real-Time Operating Systems

6.2 Real-Time Systems with Windows CE

Roadmap of Section 6.2

- Windows CE overview
- Windows CE scheduling + memory management
- Windows CE interrupt architecture
- Deterministic real-time systems with Windows CE

Definition of a Real-Time System

- From comp.realtime:

  "A real-time system is one in which the correctness of the computations not only depends on the logical correctness of the computation, but also on the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred."

- The RT OS is just one element of the complete real-time system and must provide sufficient functionality to enable the overall real-time system to meet its requirements.
  - Distinguish between a fast operating system and an RTOS

Requirements for a RT OS

- The OS (operating system) must be multithreaded and preemptive
- The OS must support thread priority
- A system of priority inheritance must exist
- The OS must support predictable thread synchronization mechanisms

In addition, the OS behavior must be predictable. This means real-time system developers must have detailed information about the system interrupt levels, system calls, and timing:

- The maximum time during which interrupts are masked by the OS and by device drivers must be known.
- The maximum time that device drivers use to process an interrupt, and specific IRQ information relating to those device drivers, must be known.
- The interrupt latency (the time from interrupt to task run) must be predictable and compatible with application requirements.

Real-Time Systems with Windows CE

- High-performance embedded applications must often manage time-critical responses.
  - manufacturing process controls,
  - high-speed data acquisition devices,
  - medical monitoring equipment,
  - laboratory experiment control,
  - automobile engine control,
  - robotics systems.
- Validating such an application means examining not only its computational accuracy, but also the timeliness of its results.
- The application must deliver its responses within specified time parameters in real-time.

Windows CE Characteristics

CE kernel design meets the minimum requirements of an RTOS:

- multithreaded and preemptive.
- supports 256 levels of thread priority.
- supports a system of priority inheritance (to correct priority inversion)
- predictable thread synchronization mechanisms,
  - including such wait objects as mutex, critical section,
  - named and unnamed event objects, which are queued based on thread priority.
  - Windows CE supports access to system timers.

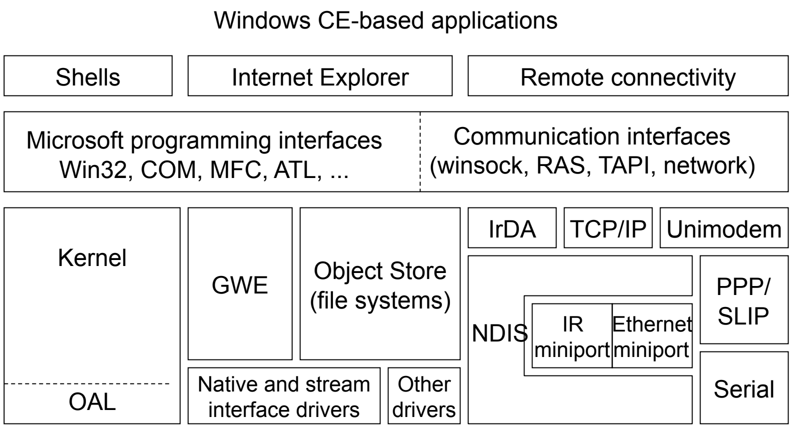
- Interrupt latency is predictable and bounded.
- The time for every system call (KCALL) is predictable and independent of the number of objects in the system.
  - The system call time can be validated using the instrumented kernel

Windows CE Modules

Windows CE has four primary modules or groups of modules.

- The kernel supports basic services
  - process and thread handling
  - memory management.
- The file system supports persistent storage of information.
- The Graphics, Windowing, and Events Subsystem (GWE)
  - controls graphics and window-related features.
- The communications interface supports the exchange of information with other devices.
- Additional modules
  - managing installable device drivers
  - supporting COM/OLE

Windows CE System Architecture (5.0 & earlier)



Kernel

- Core of the operating system; file: Nk.exe
- Windows CE kernel implements:
  - Scheduling, thread synchronization
  - Processing of exceptions and interrupts
  - Virtual memory management
- Supports execution in place (XIP) from ROM
  - Demand paging into program memory
- Portable
  - 32-bit, little endian processors
  - Support of translation look-aside buffer (TLB) for virtual to physical address mapping

Scheduling

- Win32 process and thread model
  - Round-robin, priority-based scheduler
  - 32 processes, unlimited number of threads
  - 8 priorities (256 in Win CE 3.0); circular run queue per priority
  - Scheduler operates on 25 ms quantum; adjustable
  - Thread priorities are fixed, no aging
  - Priority inheritance protocol for critical sections
- Synchronization:
  - Wait functions (WaitForSingleObject()...)
  - Mutex objects
  - Event objects
  - Semaphores to be supported in future releases

Threads and Thread Priority

- 32 simultaneous processes; one primary thread.
  - unspecified number of additional threads.
  - actual number of threads is limited only by available system resources.
- priority-based time-slice algorithm
  - schedule the execution of threads
  - eight discrete priority levels, from 0 through 7,
  - 0 represents the highest priority (header file winbase.h)

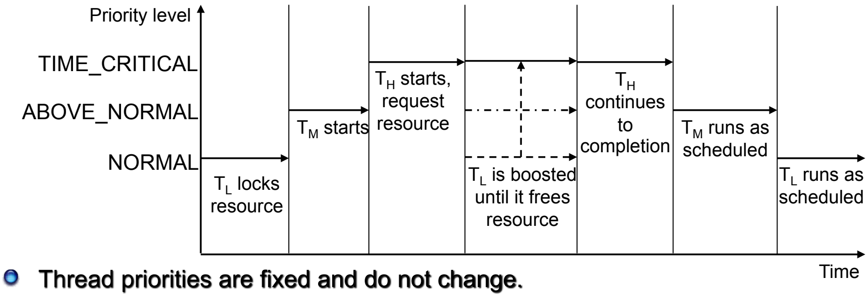
Windows CE 3.0 and later provide 256 priority levels

Priority level	Constant and Description
0 (highest)	THREAD_PRIORITY_TIME_CRITICAL (highest priority)
1	THREAD_PRIORITY_HIGHEST
2	THREAD_PRIORITY_ABOVE_NORMAL
3	THREAD_PRIORITY_NORMAL
4	THREAD_PRIORITY_BELOW_NORMAL
5	THREAD_PRIORITY_LOWEST
6	THREAD_PRIORITY_ABOVE_IDLE
7 (lowest)	THREAD_PRIORITY_IDLE (lowest priority)

Priority Assignment

- Levels 0 and 1: real-time processing and device drivers;
- Levels 2-4: kernel threads and normal applications;
- Levels 5-7: apps that can always be preempted by other apps.
- Preemption is based solely on the thread's priority.
  - Threads with a higher priority are scheduled to run first.
  - Threads at the same priority level run in a round-robin fashion with each thread receiving a quantum or slice of execution time.
  - The quantum has a default value of 25 milliseconds (CE version 3.0 and later supports changes to the quantum value).
  - Threads at a lower priority do not run until all threads with a higher priority have finished, that is, until they either yield or are blocked.
  - Exception: threads at the highest priority level (level 0) do not share the time slice with other threads at the highest priority level. These threads continue executing until they have finished.
- Thread priorities are fixed and do not change.
  - Windows CE does not age priorities and does not mask interrupts based on these levels

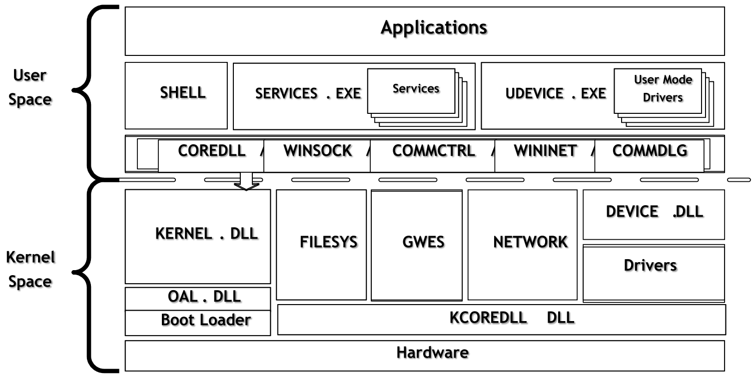
Priority Inheritance – circumvent priority inversion problems



- Thread priorities are fixed and do not change.
- Windows CE does not age priorities and does not mask interrupts based on these levels.
- Only kernel modifies priorities temporarily to avoid "priority inversion."

Thread Synchronization

- CE offers a rich set of "wait objects" for thread synchronization.
  - critical section, event, and mutex objects.
  - wait objects allow a thread to block its own execution and wait until the specified object changes.
- Windows CE queues mutex, critical section, and event requests in "FIFO-by-priority" order
  - a different FIFO queue is defined for each of the eight discrete priority levels.
  - A new request from a thread at a given priority is placed at the end of that priority's list.
  - The scheduler adjusts these queues when priority inversions occur.
- Windows CE supports standard Windows timer API functions
  - Obtain time intervals from the kernel through software interrupts.
  - Threads can use the system's interval timer by calling **GetTickCount**, which returns a count of milliseconds.
  - Use **QueryPerformanceCounter** and **QueryPerformanceFrequency** for more detailed timing information.
    (OEM must provide higher-resolution timer and OAL interfaces to the timer.)



Windows Embedded CE 6.0 Architecture.

HPI Embedded Systems Programming

CE 6.0 User Processes

- Shell – Standard or custom interface for device
- Services.exe hosts *n* number of services
- UDevice.exe hosts *n* number of user mode drivers

HPI Embedded Systems Programming

CE 6.0 Kernel

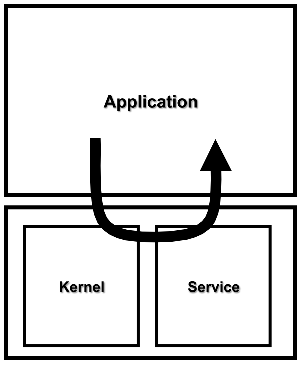
- FilesSys.dll provides file system support and communicates with file system drivers (FSD)
- GWES.dll is the Graphics, Windowing, and Events Subsystems
- Networking DLLs Networking services
- Device.dll provides device driver services
- Kernel provides basic OS services
- API calls use KCOREDLL.dll to get to other kernel services

HPI Embedded Systems Programming

Application Programs in CE

- CE Supports C/C++/C# & Threads
- Uses a critical subset of the Desktop Windows APIs – around 2,000 vs. 20,000
- Means CE application source code can be recompiled and run on the Desktop Windows OS, but the reverse is not true
- Sample browser, Media player, WordPad applications included with OS.

HPI Embedded Systems Programming



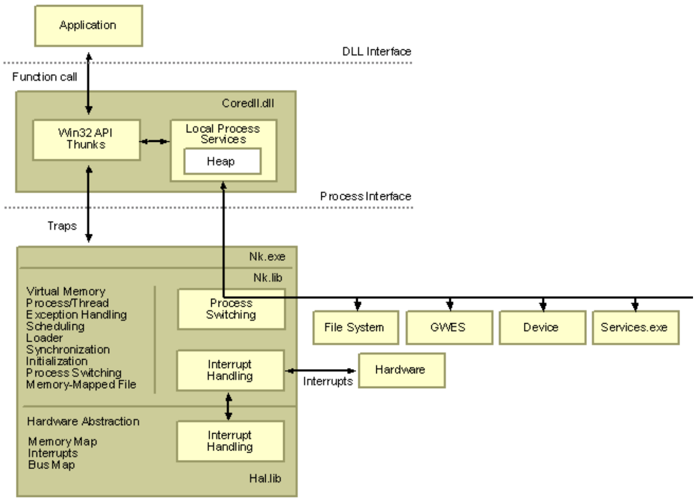
CE 6.0 System Calls

Embedded Systems Programming

Kernel (CE 6.0 & later)

- New Kernel (NK.bin) module
- Core of the operating system
- Base level functions in kernel: process, thread, and memory management
- Includes some file management functions
- Kernel services allow applications to use the core functions

Embedded Systems Programming



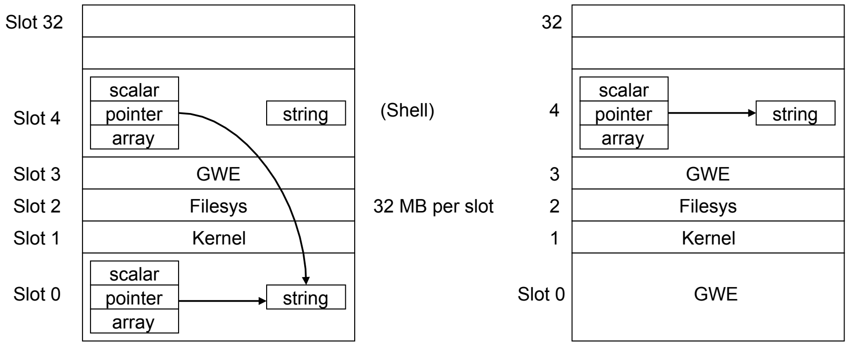
Windows CE 6.0 Kernel Architecture

Virtual Memory: Slot Model
(CE 5.0 & earlier)

- 32 MB per process virtual memory
  - Target devices: < 4Mb RAM, < 4Mb ROM
  - 2 GB address space sliced into 32 Mb slots
- One slot per process:
  - Slot is broken into 512 64kb blocks
  - Blocks are broken into 1kb/4kb pages (– depending on system)
- Assignment of slots:
  - Slot 0: active process
  - Slot 1: kernel
  - Slot 2: GWE (Graphics, Window Manager, Event Manager)
  - Slot 3: Filesys...
  - Slot 4: Shell
  - New processes get lowest available slot

Virtual Memory (contd.)

32 slots + active slot (0)



- Kernel manipulates pointer on context change
- GWE may access Shell data at correct location without copy

Virtual Memory & Real-Time

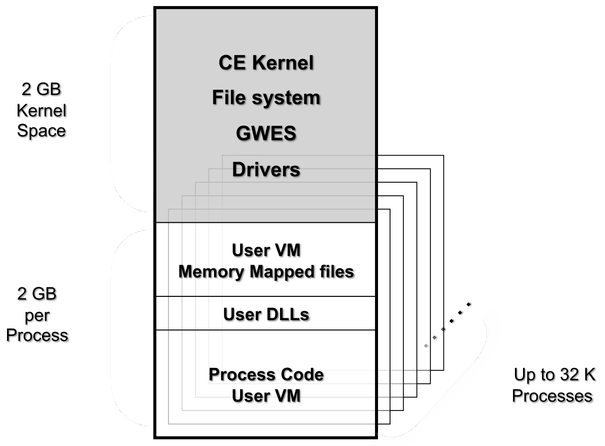
- Paging I/O occurs at a lower priority level than the real-time priority process levels.
  - Paging within the real-time process is still free to occur
  - Background virtual memory management won't interfere with processing at real-time priorities.
- Real-time threads should be locked into memory to prevent nondeterministic paging delays resulting from VM system.
- Windows CE allows memory mapping
  - Multiple processes may share the same physical memory.
  - Very fast data transfers between processes / driver / app.
  - Memory mapping can be used to dramatically enhance real-time performance

Persistent Storage

The file system supports persistent storage of information.

It includes:

- Support for file allocation table (FAT) file systems with up to nine FAT volumes.
- Transactioned file handling to protect against data loss.
- Demand paging for devices that support paging.
- FAT file system mirroring to allow preservation of the file system if power is lost or cold reset is needed.
- Installable block device drivers.



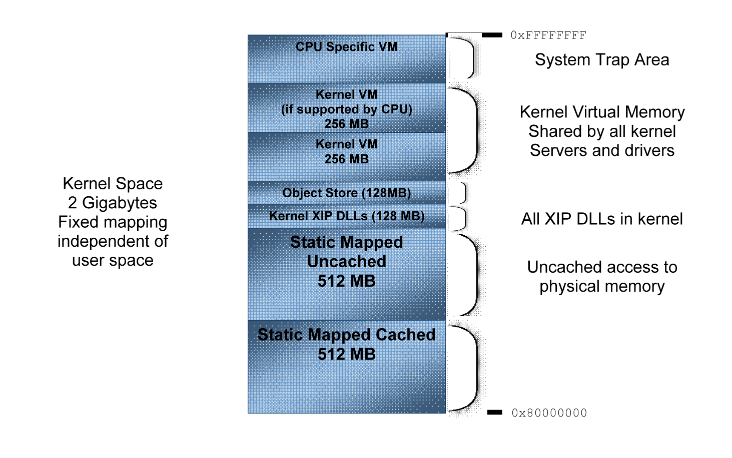
The Windows Embedded CE 6.0 Virtual Memory Space Model.

Memory Setup In CE 6.0

- 4GB 32-bit Virtual Memory Address Space
- 2GB User Space in Virtual Memory Address Space
- 2GB Kernel Space in Virtual Memory Address Space

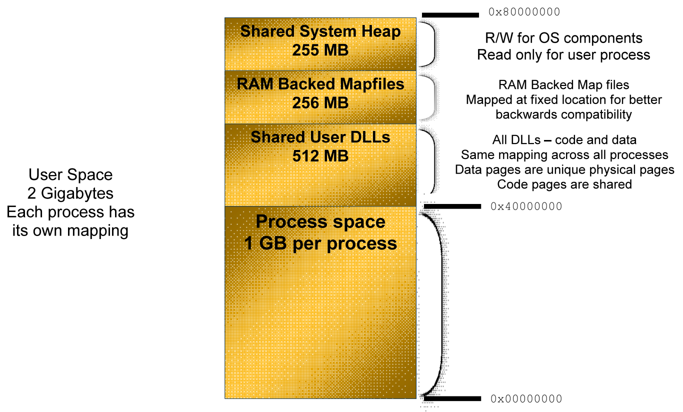
NOTE: Remember that Physical memory size is independent of the Virtual Memory Address Space!

Embedded Systems Programming



Layout of kernel space virtual memory

Embedded Systems Programming

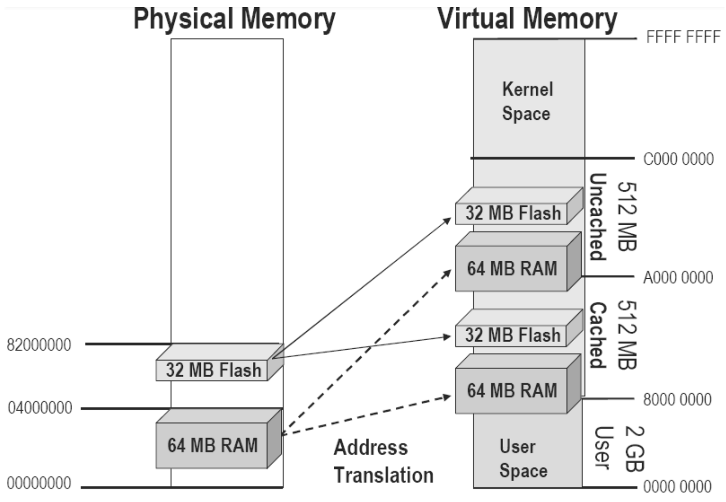


Layout of user space virtual memory

Table 6.1 CE 6.0 Virtual Memory Map.

Mode	Range	Size	Description	Comments
Kernel	0xF0000000 - 0xFFFFFFFF	256 MB	CPU specific VM	System call trap area. Kernel data page.
Kernel	0xE0000000 - 0xEFFFFFFF	256 MB	Kernel VM, CPU dependent	Kernel space virtual memory, unless disallowed by the CPU, such as SHx.
Kernel	0xD0000000 - 0xDFFFFFFF	256 MB	Kernel VM	Kernel space virtual memory, shared by all servers and drivers loaded in kernel.
Kernel	0xC8000000 - 0xCFFFFFFF	128 MB	Object store	RAM based storage for RAM file system, CEDB databases, and RAM-based registry. Legacy data store.
Kernel	0xC0000000 - 0xC7FFFFFF	128 MB	Kernel XIP DLLs	XIP DLLs for the kernel and all servers and drivers loaded in the kernel.
Kernel	0xA0000000 - 0xBFFFFFFF	512 MB	Statically mapped Uncached	Direct access to physical memory bypassing the CPU cache.
Kernel	0x80000000 - 0x9FFFFFFF	512 MB	Statically mapped Cached	Direct access to physical memory accessed through the CPU cache.
User	0x7FF00000 - 0x7FFFFFFF	1 MB	Unmapped for protection	Buffer between user and kernel spaces.
User	0x70000000 - 0x7FEFFFFF	255 MB	Shared system heap	Shared heap between the kernel and the process. Kernel and kernel servers can allocate memory in it and write to it. Read only for user processes It is a system optimization that allows a process to get data from a server without having to make a kernel call.

User	0x60000000 - 0x6FFFFFFF	256 MB	RAM backed map files	RAM backed mapfiles are mapped at fixed location for backward compatibility. RAM backed map files are memory mapped file objects that do not have an actual file underneath them. They are acquired by calling *CreateFileMapping* with *hFile* equal to INVALID_HANDLE_VALUE. This region provides backward compatibility for applications that used RAM-backed map files for cross-process communication, expecting all processes to map views at the same virtual address.
User	0x40000000 - 0x5FFFFFFF	512 MB	User mode DLLs Code and data	DLLs loaded at bottom and grow up: • Based starting at 0x40000000. • Code and data are intermixed. • A DLL loaded in multiple processes will load at the same address in all processes. • Code pages share same physical pages. • Data pages have unique physical pages for each process.
User	0x00010000 - 0x3FFFFFFF	1 GB	Process User allocatable VM	Executable code and data User VM (heap) virtual allocations: 1 VM allocations start above the exe and grow up.
User	0x00000000 - 0x00010000	64 KB	CPU dependent user kernel data	User kernel data is always r/o for user. Depending on CPU, it can be kernel r/w (ARM), or kernel r/o (all others).



Virtual to physical memory mapping on a device (example)

What is in Memory?

- OS Kernel
- Application Code & Data
- Object Store – File System, Registry, Built-in Compact Data Base
- Memory Mapped Files

Communications Interface

- Support for serial communications, including infrared links.
- Support for Internet client applications,
  - including Hypertext Transfer Protocol (HTTP) and
  - File Transfer Protocol (FTP) protocols.
- A Common Internet File System (CIFS) redirector for access to remote file systems by means of the Internet.
- A subset of Windows Sockets (Winsock) version 1.1
  - support for Secure Sockets.
- A TCP/IP transport layer configurable for wireless networking.
  - An Infrared Data Association (IrDA) transport layer for robust infrared comm.
  - Point-to-Point Protocol (PPP) and Serial Line Internet Protocol (SLIP) for serial-link networking.
- Networking through network driver interface specification (NDIS).
  - Support for managing phone connections with the Telephony API (TAPI).
  - Remote Access Service client for connections to remote file systems by modem.

Graphics, Windowing, and Events Subsystem (GWE)

The GWE module supports the graphics and windowing functionality.

It includes:

- Support for a broad range of window styles, including overlapping windows.
- A large selection of customizable controls.
- Support for keyboard and stylus input.
- A command bar combining the functionality of a toolbar and a menu bar.
- An **Out of Memory** dialog box that requests user action when the system is low on memory.
- Full UNICODE support.

Graphics Device Interface (GDI)

- A multiplatform graphics device interface (GDI) that supports the following features:
- Both color and grayscale displays, with color depths of up to 32 bits per pixel.
- Palette management.
- TrueType and raster fonts.
- Printer, memory, and display device contexts.
- Advanced shape drawing and bit block transfer capabilities.

Device Drivers

- Device drivers in user-mode processes
- Only small part of driver is linked with kernel
  - Keep interrupt service routines short
- No nestable interrupts
  - All interrupts masked in service routine
- Drivers can be layered

Interrupt Handling: IRQs, ISRs, and ISTs

- Windows CE balances performance and ease of implementation by splitting interrupt processing into two steps: an interrupt service routine (ISR) and an interrupt service thread (IST).
- Hardware interrupt request lines (IRQ) are associated with ISRs.
  - When interrupts are enabled and an interrupt occurs, the kernel calls the registered ISR for that interrupt.
  - It is ISR's responsibility to direct the kernel to launch the appropriate IST.
- ISR performs minimal processing and returns an interrupt ID to the kernel.
- The kernel examines interrupt ID and sets the associated event.
- The interrupt service thread is waiting on that event.
  - When the kernel sets the event, the IST starts its additional interrupt processing.
  - Most of the interrupt handling actually occurs within the IST.
  - The two highest thread priority levels (levels 0 and 1) are usually assigned to ISTs.

Windows CE Interrupt Architecture - Nested interrupts

- Full support for nested interrupts
- Based on support by the CPU and/or additional hardware
- Nested in order of priority
- Kernel will save and restore all required registers

Interrupt Architecture

- ISR runs as part of the kernel
  - Multiple interrupt priorities dependent on CPU and available hardware
  - Can't make system calls while in ISR
    - No memory allocation, file system access, load module, etc.
- IST runs as part of a user mode DLL
  - Full access to system services
  - Can still access hardware if necessary
  - Utilizes normal thread priorities and scheduler
  - ISR and IST priorities independent for maximum flexibility

ISR and IST Model

- Interrupt Service Routine
  - Typically very short, fast, assembly code
  - Job is to return logical Interrupt ID to the Kernel.
  - For Example… Serial Interrupt may be identified as SYSINTR_SERIAL

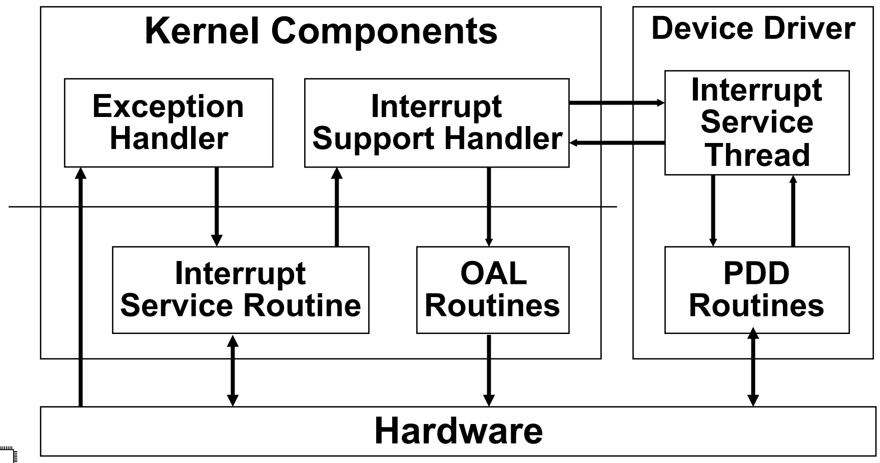
```
// ISR
// Interrupts are Disabled
Identify the Interrupt, Mask or Dismiss the Interrupt
Return the Interrupt ID
// Interrupts are on again.
```


ISR and IST Model

- Interrupt Service Thread
  - Part of a device driver (DLL)
  - Built in or loaded by Device.exe

```
// Serial Device Driver (IST)
// Setup Hardware
hEvent=CreateEvent( … );
InterruptInitialize(hEvent,SYSINTR_SERIAL);
CreateThread( … );
// ------------------ Thread Code -----------------
While( TRUE ) {
       WaitForSingleObject(hEvent,timeout);
       { DoStuff( ); }
       InterruptDone(SYSINTR_SERIAL);
}
```

Interrupt Block Diagram
Drivers for built-in devices



Windows CE: Architectural Remarks

- Windows CE runs all device drivers inside a user-space process: Devices.exe
  - Resembles microkernel architecture
- Programmer has full control on priority of Interrupt Service Threads (IST)
  - Kernel-mode Interrupt Service Routine (ISR) is short and mainly signals an event to IST
  - Windows CE can be configured to run everything in kernel mode (minimize context switching overheads)

Bounded Interrupt Latency
(for threads locked in memory)

ISR latency:

- **start of ISR = $Kernel_1$ + $d_{ISR\_Current}$ + sum(d_{ISR_Higher})**
  1. **$Kernel_1$** = latency value due to processing within the kernel.
  2. ***$d_{ISR\_Current}$*** = duration of ISR in progress at interrupt arrival. (0 .. max(T_{exec}(ISR))).
  3. **sum(d_{ISR_Higher})** = sum of the durations of all higher priority ISRs that arrive before this ISR starts; (for interrupts that arrive during the time **$Kernel_1$** + *dISR_Current*)

IST latency:

- **start of IST = $Kernel_2$ + sum(d_{IST}) + sum(d_{ISR})**
  1. **$Kernel_2$** = latency value due to processing within the kernel.
  2. **sum(d_{IST})** = sum of the durations of all higher priority ISTs and thread context switch times that occur between this ISR and its start of IST.
  3. **sum(d_{ISR})** = The sum of the durations of all other ISRs that run between this interrupt's ISR and its IST.

Example

- Embedded system with only one critical-priority ISR.
  - ISR is set to the highest priority (no higher priority ISRs) -> d_{ISR_Higher} = 0.
  - $latency_{min}$ = $Kernel_1$.
  - $latency_{max}$ = $Kernel_1$ plus the duration of the longest ISR.
- No other ISTs can intervene between ISR and its IST.
  - However, it is possible that other ISRs can be processed between the time-critical ISR and the start of its associated IST.
- Pathological case:
  - A constant stream of ISRs, postpones the start of IST indefinitely.
  - Unlikely, OEM has control over the number of interrupts in the system.
- To minimize latency times, the OEM can control the processing times of the ISR and IST, interrupt priorities, and thread priorities.

Validating the Real-time Performance of Windows CE

- In-house inspection and analysis of the kernel code by the Windows CE development team, and
- OEM and ISV (independent software vendor) timing validation of specific configurations using tools that will be provided in future versions of the Windows CE Embedded Toolkit for Visual C++.

The Windows CE Embedded Toolkit for Visual C++ includes:

- An instrumented version of the kernel for timing studies, and
- The Intrtime.exe utility for observing minimum, maximum, and average time to interrupt processing.

Performance Tools

- Provided in Platform Builder to measure real-time performance of your system
  - ISR/IST Latency
  - Scheduling performance
- Event logging tool useful for debugging and performance tuning
- More information on these tools available in the Platform Builder Online Help

Measurements – varying number of system objects

- Start of ISR times are independent of #system objects

Start of ISR_{Max}	Numbers of background threads (with one event per thread)	Background thread priority
8.4 μS	0	7
8.6 μS	5 (Note: represents only 100 tests)	7
9.0 μS	10 (Note: represents only 100 tests)	5
14.8 μS	10	5
19.2 μS	10	5
17.0 μS	10	7
12.8 μS	20	5
11.0 μS	20 (Note: represents only 100 tests)	7
10.0 μS	50	7
15.0 μS	100	5
15.6 μS	100	7

Windows CE Has Deterministic Performance!

ILTiming and OSBench tools running on development versions show that latencies are bounded

- For a Pentium 166 MHz class system
  (Remember: embedded systems are small and with limited resources - CPU, Memory, Power)
  - ISR < 10 μS
  - IST < 100 μS

Getting Real-Time Performance

- Don't:
  - Spend inordinate amounts of time in ISRs
  - Spin in your highest priority thread, you'll starve the system
  - Use APIs that are not real-time and expect real-time performance
    - SetTimer, file system calls, process or thread creation,…
  - Allow priority inversions to occur

Getting Real-Time Performance

- Do:
  - Pre-allocate all your resources
    - Memory, threads, processes, mutexes, semaphores, events, etc…
  - Buffer data in ISR if passing it directly to the IST isn't fast enough
  - Use ISR to do all work if…
    - …No system services are required
    - …No extensive processing (long ISR time) required
  - Set priorities and quantums correctly
  - Use LoadDriver() to instead of LoadLibrary() to avoid page faults
    - Or turn the demand-pager off

References

- msdn.microsoft.com/embedded/usewinemb/ce/techno/realtme/default.aspx
- msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/cmconreal-timeperformancefunctionality.asp
- msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/realtimecapabilities.asp
- msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/windowsce.asp

Further Reading

- Douglas Boling, Programming Microsoft Windows CE .NET, Third Edition, MS Press, 2003.
- msdn.microsoft.com/embedded/windowsce/default.aspx