Windows NT and Real-Time?

- Reading: “Inside Microsoft Windows 2000”, (Solomon, Russinovich, Microsoft Programming Series)
- “Real-Time Systems and Microsoft Windows NT” (MSDN Library)

Priorities in Windows NT/2000

Priority Spectrum

Real-time classes

Dynamic classes

Real-time time-critical

Real-time not real

System-level like input, cursor, cache flushing, file sys., drivers

Real-time idle

Normal time/critical

High foreground

Normal foreground

Low foreground/background

Normal time/file

Idle thread
IO System Components (Windows 2k)

Device Driver Layering
Device Driver Layering (2)

Primary Device Driver Routines

- NT/2000 device drivers run entirely within the system process and have access to all hardware through the HAL. A typical device driver will have several components:
Control Flow for an IO Operation

1. Call `ReadFile()`
2. Call `NtReadFile()`
3. `INT 2E`
4. `return to caller`

Whether to wait depends on overlapped flag

Queueing and Completing a Synchronous Request

1. I/O request passes through subsystem DLL
2. `NtWriteFile(file_handle, ..., char_buffer)`
3. Create IRP and send it to device driver
4. Handle interrupt and return success or error status
5. Complete IRP and return success or error status
6. Perform I/O and Interrupt
7. Transfer data specified in IRP
Servicing a Device Interrupt (only Phase I)

1. The device interrupt for service.

2. The ISR stops the device interrupt and queues a DPC.

3. The kernel's interrupt dispatcher transfers control to the device's service routine.

Servicing a Device Interrupt (Phase II)

1. The device IRQs, rings, and DPC processing occurs.

2. The DPC routine starts the first IO request in the device queue and then completes interrupt servicing.

3. The interrupt dispatcher transfers control to the driver's DPC routine.
Completing an I/O Request (Phase I)

1. The DPC routine calls the I/O manager to complete the original I/O request.
2. The I/O manager queues an APC to complete the I/O request within the caller's context.

Completing an I/O Request (Phase II)

1. The kernel-mode APC routine writes data to the thread's address space, sets the original file handle to the suspended state, queues any user-mode APCs for execution, and deposes the IRP.
2. The interrupt dispatcher transfers control to the I/O manager's APC routine.

Thread's APC queue

I/O manager

Device driver

Dispatch routine(s)
Start I/O
ISR
DPC routine(s)

Environment subsystem or DLL

User mode

Kernel mode

IRP

APC

IRP

IRP

Thread's APC queue

IRP

APC

IRP

APC

IRP

IRP

APC

IRP

IRP
Priority Levels vs. Interrupt Levels

- The HAL maps hardware-interrupt numbers to IRQLs.
- IRQLs are not the same as IRQs in x86.
- Scheduling priority is attribute of thread, while IRQL is attribute of an interrupt source.
- Lazy IRQL management for slow PICs.
- Code running at DPC/dispatch level or above can’t wait on object if so would necessitate scheduler to invoke another thread.

Memory Management

- 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, but this really ensures that background virtual memory management won’t interfere with processing at real-time priorities.
- Windows NT permits an application to lock itself into memory so that it is not affected by paging within its own process. This allows even very large processes (such as raster image processing, where some processes are over 100MB) to lock all their memory down into physical memory and avoid the overhead of paging, while allowing the rest of the system to function normally.
- Windows NT memory management allows for memory mapping, which permits multiple processes—even device drivers and user applications—to share the same physical memory. This results in very fast data transfers between cooperating processes or between a driver and an application. Memory mapping can be used to dramatically enhance real-time performance.
Windows 2000/NT and Real-Time Processing

- Windows 2000/NT does not prioritize device IRQs in controllable way.
- User-level applications execute only when a processor’s IRQL is at passive level.
- System’s devices and device drivers – not the OS – ultimately determine the worst-case delay.
- This is a problem with off-the-shelf hardware and drivers.
- System designer must bound the length of device’s ISR and DPC in the worst case.
- Embedded versions of Windows NT/2000 provide control over memory footprint etc, but are not real-time capable.
- Extensions of real-time kernels can be provided through custom extensions of the HAL.

VenturCom RTX Architecture

Figure 1: RTX Architecture