A technical text – computer science.

Excerpt from the page 85 Mark Russinovich

(Windows Internals – System Mechanisms)

The Windows operating system provides several base mechanisms that kernel-mode components such as the executive, the kernel, and device drivers use.

This chapter explains the following system mechanisms:

- Trap dispatching, including interrupts, deferred procedure calls (DPCs), asynchronous procedure calls (APCs), exception dispatching, and system service dispatching.
- The executive object manager.
- Synchronization, including spinlocks, kernel dispatcher objects, how waits are implemented, as well as user-mode-specific synchronization primitives that avoid trips to kernel mode.
- System worker threads.
- Miscellaneous mechanisms such as Windows global flags.
- Advanced local procedure calls (ALPCs).
- Kernel Event Tracing.
- Wow64.
- User-mode debugging.
- The image loader.
- Hypervisor (Hyper-V).
- Kernel Transaction Manager (KTM).
- Kernel Patch Protection (KPP).
- Code integrity.

TRAP DISPATCHING

Interrupts and exceptions are operating system conditions that divert the processor to code outside the normal flow of control. Either hardware or software can detect them.

The term trap refers to a processor's mechanism for capturing an executing thread when an exception or interrupt occurs and transferring control to a fixed location in the operating system.

In Windows, the processor transfers control to a trap handler, which is a function specific to a particular interrupt or exception.

The kernel distinguishes interrupts and exceptions in the following way:

An interrupt is an asynchronous event (one that can occur at any time) unrelated to what the processor is currently executing.

Interrupts are generated primarily by:

- I/O devices
- Processor clocks
- Timers

They can be enabled or disabled.

An exception, by contrast, is a synchronous condition resulting from execution of a particular instruction.

Running a program again with the same data under the same conditions can reproduce exceptions.

Examples of exceptions include:

- Memory access violations
- Certain debugger instructions
- Divide-by-zero errors

The kernel also treats system service calls as exceptions, although technically they are system traps.

Either hardware or software can generate exceptions and interrupts.

Examples:

- A bus error exception is caused by a hardware problem.
- A divide-by-zero exception is caused by a software bug.
- An I/O device can generate an interrupt.
- The kernel itself can issue a software interrupt, such as an APC or DPC.

When a hardware exception or interrupt occurs, the processor records enough machine state on the kernel stack of the interrupted thread so that execution can later continue as if nothing happened.

If the thread was executing in user mode, Windows switches to the thread's kernel-mode stack.

Windows then creates a trap frame on the kernel stack of the interrupted thread.

The trap frame stores the execution state of the thread.

It is a subset of the thread's full context.

INTERRUPT DISPATCHING

Hardware-generated interrupts typically originate from I/O devices that must notify the processor when they need service.

Interrupt-driven devices allow the operating system to maximize processor use by overlapping computation with I/O operations.

Typical process:

1. A thread starts an I/O transfer.
2. The thread continues other useful work.
3. The device completes the transfer.
4. The device interrupts the processor for service.

Interrupt-driven devices include:

- Pointing devices
- Printers
- Keyboards
- Disk drives
- Network cards

System software can also generate interrupts.

For example:

- The kernel can issue a software interrupt to initiate thread dispatching.
- The kernel can asynchronously break into execution of a thread.

The kernel can disable interrupts temporarily, but only at critical moments, such as:

- While processing another interrupt
- While dispatching an exception

The kernel installs interrupt trap handlers to respond to device interrupts.

These handlers transfer control either to:

- An external interrupt service routine (ISR) supplied by a device driver, or
- An internal kernel routine.

Device drivers provide ISRs for servicing hardware interrupts.

HARDWARE INTERRUPT PROCESSING

On supported Windows hardware platforms, external I/O interrupts arrive on one of the interrupt controller's lines.

The controller then interrupts the processor through a single line.

Once interrupted, the processor queries the controller to obtain the interrupt request (IRQ).

The interrupt controller translates the IRQ into an interrupt number.

This number is used as an index into the Interrupt Dispatch Table (IDT).

Control is then transferred to the corresponding interrupt dispatch routine.

At system boot time, Windows fills the IDT with pointers to kernel routines that handle interrupts and exceptions.

IRQLs (INTERRUPT REQUEST LEVELS)

Although hardware interrupt controllers prioritize interrupts to some extent, Windows imposes its own priority scheme called IRQLs.

The kernel represents IRQLs internally:

- x86: 0 through 31
- x64 / IA64: 0 through 15

Higher numbers mean higher priority.

Interrupts are serviced in priority order.

A higher-priority interrupt can preempt a lower-priority interrupt already being serviced.

When an interrupt occurs:

1. The processor saves the interrupted thread's state.
2. The trap dispatcher raises the IRQL.
3. The dispatcher calls the interrupt service routine.
4. After execution, the IRQL is lowered.
5. The saved machine state is restored.
6. The interrupted thread resumes.

IRQLs are different from thread priorities.

- Thread priority is an attribute of a thread.
- IRQL is an attribute of an interrupt source.

Each processor also has a current IRQL setting.

That setting determines which interrupts it can receive.

As a kernel-mode thread runs, it may raise or lower IRQL directly or indirectly while accessing synchronization objects.

Interrupts with IRQL above the current level are delivered immediately.

Interrupts with equal or lower IRQL are masked until the level is lowere

COMPACTNESS AND THE HEINE–BOREL THEOREM
Strana 115

4.4.3 UNIFORM CONTINUITY ON CLOSED BOUNDED INTERVALS

Using the Heine–Borel Theorem, it can be shown that every continuous function on a closed bounded interval is uniformly continuous on that interval.

The basic idea is the following:

If f is continuous on the closed bounded interval [a, b], then for every $\varepsilon > 0$ and for every point x in [a, b], there exists a $\delta > 0$ such that whenever y belongs to the interval:

$(x - \delta, x + \delta)$

it follows that:

$|f(x) - f(y)| < \varepsilon$

Thus, around every point x in [a, b], there exists an open interval having the desired continuity property.

The Heine–Borel Theorem states that the closed bounded interval [a, b] is compact.

Therefore, every open cover of [a, b] has a finite subcover.

This means that the infinitely many open intervals centered at each point x can be reduced to finitely many intervals covering [a, b].

Since each of these finitely many intervals has an associated positive δ, one may try to choose the smallest δ and use it in proving uniform continuity.

FIRST SUBTLETY

There is a complication.

From continuity at x, one only knows that:

$|f(y) - f(x)| < \varepsilon$

for y near x.

However, uniform continuity requires showing:

$|f(y) - f(z)| < \varepsilon$

for any two nearby points y and z.

So the estimate must compare arbitrary nearby points, not only points with x.

STANDARD ANALYSIS TRICK

To solve this, instead of using ε, use:

$\varepsilon / 2$

Because continuity works for every positive number, it also works for ε/2.

So choose $\delta > 0$ such that:

$|f(y) - f(x)| < \varepsilon/2$

whenever y lies in:

$(x - \delta, x + \delta)$

Then if both y and z are in that interval, the triangle inequality gives:

$|f(y) - f(z)| \leq |f(y) - f(x)| + |f(z) - f(x)|$

Hence:

$|f(y) - f(z)| < \varepsilon/2 + \varepsilon/2 = \varepsilon$

So the first difficulty is resolved.

SECOND SUBTLETY

Another problem remains.

If y and z are close together, how do we know they both lie in the same interval of the finite open cover?

Even though [a, b] is covered by finitely many open intervals, two nearby numbers do not automatically belong to the same one.

SOLUTION USING ENDPOINTS

Consider all endpoints of the finitely many open intervals in the finite cover.

Because the cover is finite, there are only finitely many such endpoints.

Choose δ not as the minimum of the interval radii, but as the smallest distance between any two distinct endpoints.

Then if:

$|y - z| < \delta$

there can be at most one endpoint between y and z.

This guarantees that y and z must both lie inside one of the covering intervals.

WHY THIS WORKS

Intervals in an open cover must overlap sufficiently to cover the whole closed interval [a, b].

Therefore, an endpoint of one interval must lie inside another interval.

So if y and z are very close and straddle at most one endpoint, they still remain together inside some open interval of the finite cover.

CONCLUSION

Using compactness from the Heine–Borel Theorem:

- every point has a local continuity neighborhood
- finitely many such neighborhoods cover [a, b]
- one common δ can be chosen
- therefore continuity becomes uniform continuity

FINAL RESULT

Every continuous function on a closed bounded interval [a, b] is uniformly continuous on that interval.

The material is taken from Jonathan M. Kane, Writing Proofs in Analysis, pages 115–116.

EXCERPT FROM THE STORY
DOWN AND OUT IN THE YEAR 2000
Page 98

By Kim Stanley Robinson

From the book Cyberpunk
Edited by Victoria Blake

DIALOGUE IN A COLLAPSING URBAN LANDSCAPE

“What you doing here?”

“Sitting!”

The man was startled and nervous.

“Just sitting in a park!”

“This ain’t no park, man.

This is our front yard.

You see any front yard to these apartment buildings here?

No.

This here is our front yard, and we don’t like people just coming into it and sitting down anywhere!”

The man stood and walked away.

He looked back once.

His expression was angry and frightened.

The other man sitting on the park benches looked at Lee curiously.

TWO DAYS LATER

Lee was nearly out of money.

He walked over to Connecticut Avenue, where his old friend Victor played harmonica for coins whenever he could not find other work.

Today Victor was there, performing:

Amazing Grace

He stopped when he saw Lee.

“Robbie! What’s happening?”

“Not much. You?”

Victor gestured toward his empty hat lying on the sidewalk before him.

“You see it.

Don’t even have seed coin for the cap, man.”

“So you ain’t been getting any gardening work lately?”

“No, no. Not lately.

I do all right here, though.

People still pay for music, man, some of them.

Music’s the angle.”

MEMORY OF FORMER WORK

Victor looked at Lee, his face tightened against the sun.

They had once worked together for the park service.

During summer mornings they drove a truck through city streets.

They stopped at every tree.

One of them would hoist the other upward in slings.

The man raised into the branches had to balance like an acrobat.

He moved carefully while cutting every branch below twelve feet.

The chainsaw had to be handled precisely to avoid cutting legs or causing injury.

Those had been good times.

But now the park service was gone.

Victor sat behind an empty hat and looked at Lee with stoic resignation.

NATURE RETURNS

“Do you ever look up at the trees anymore, Robbie?”

“Not much.”

“I do.

They’re growing wild, man.

Every summer they go like crazy.

Pretty soon people are gonna have to drive their cars through the branches.

The streets’ll be tunnels.

And with half the buildings in this area falling down...

I like the idea that the forest is taking this city back again.

Running over it like kudzu.

Till maybe it just be forest again at last