Windows 2000 - Interrupts

„Ausgewählte Betriebssysteme“

Institut Betriebssysteme
Fakultät Informatik

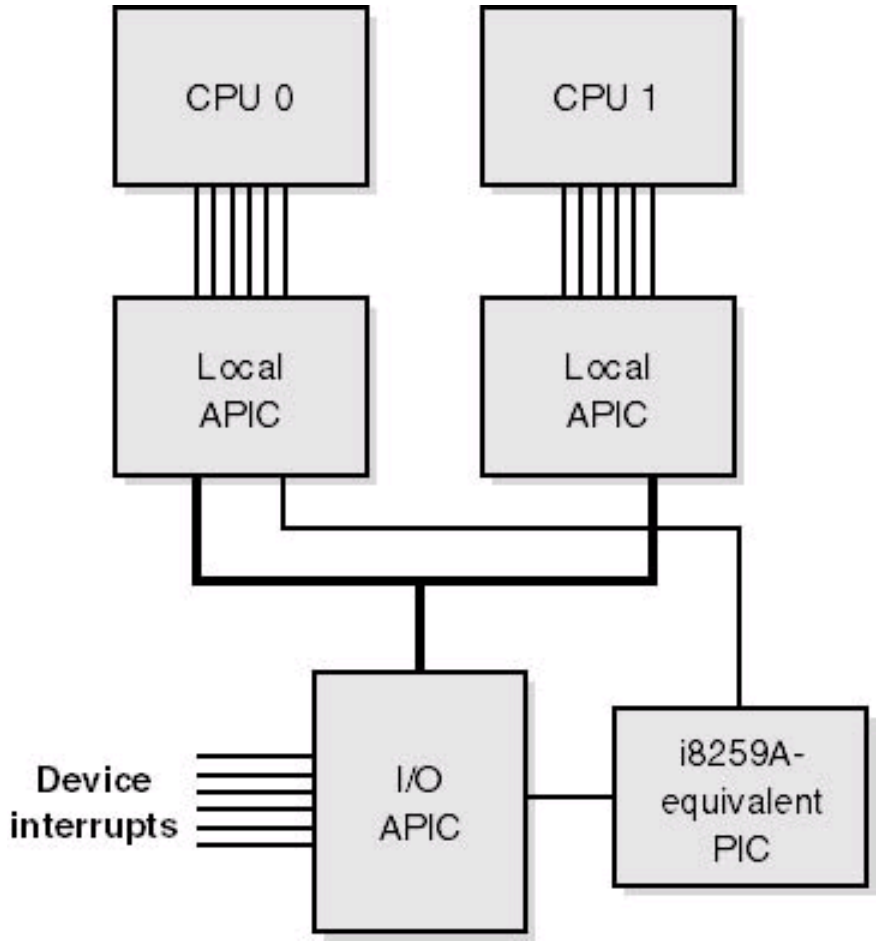
Interrupts

- Software and Hardware Interrupts and Exceptions
- Kernel installs interrupt trap handlers
- Interrupt trap handler transfers control to:
  - External routine (Interrupt Service Routine)
  - Internal kernel routine that responds to interrupt
- ISR supplied by device driver to service device interrupts
- Kernel provides routines for other types of interrupts (e.g. timer)

Hardware Interrupt Processing

- External I/O interrupts come into one of PIC lines
- PIC interrupts processor on single line
- Processor queries PIC to get interrupt request (IRQ)
- PIC translates IRQ to number
- Number is used as index into interrupt dispatch table
- Control transferred to interrupt dispatch routine
- Windows 2000 fills IDT at boot time with kernel routines
- IDT also used for exception handling (e.g. PF at 0xe)

PIC/APIC



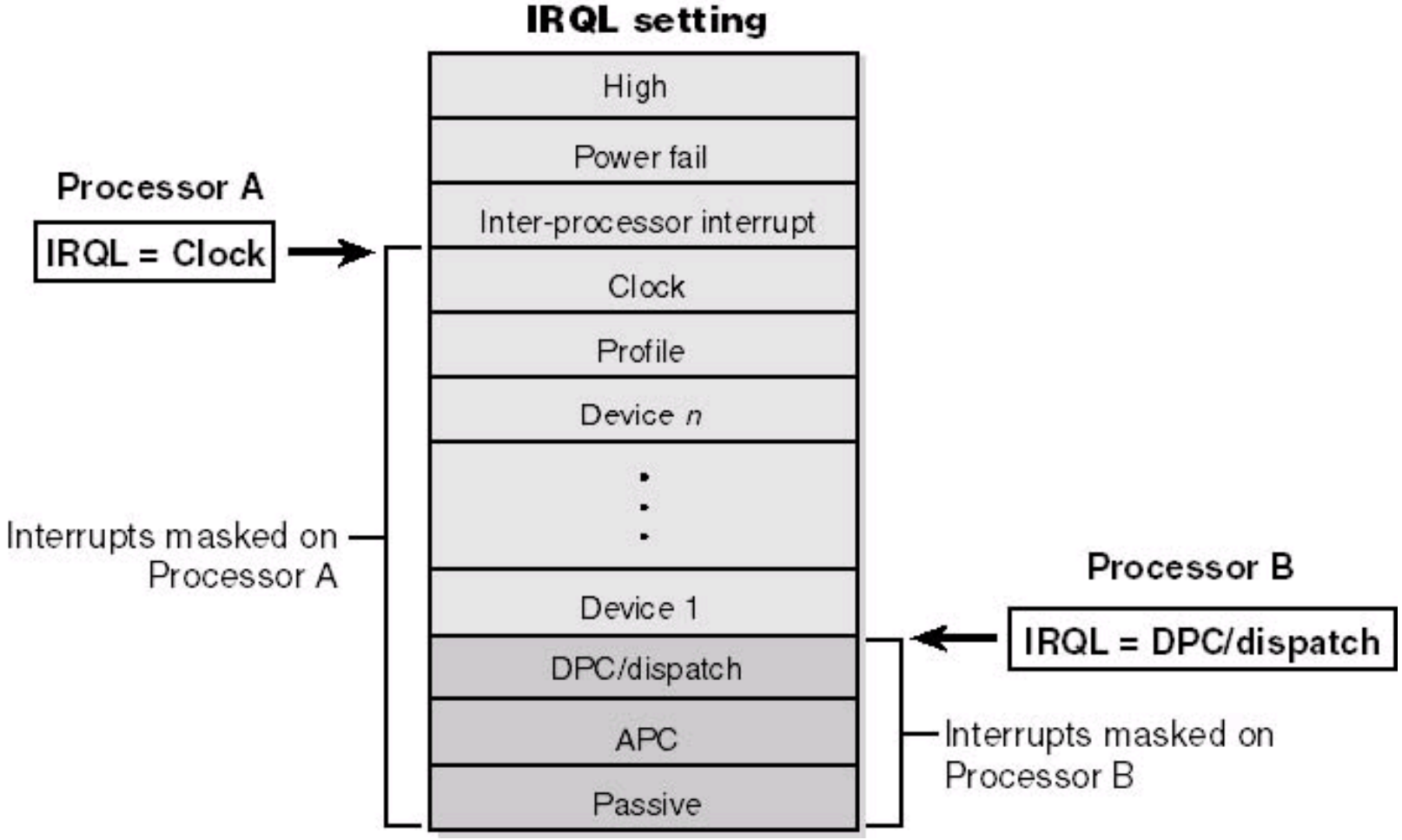
PIC/APIC (2)

- PIC has 15 interrupt lines
- APIC has 256 interrupt lines
- APIC supports PIC compatibility mode
- APIC consists of:
  - I/O APIC receives interrupt from device
  - Local APIC receives interrupt from I/O APIC
  - Local APIC interrupts CPU
  - i8259A compatible IC translates APIC signals to PIC signals
- I/O APIC responsible for interrupt routing
- Routing is software selectable (done by HAL)

Software Interrupt Request Levels (IRQLs)

- Windows 2000 own interrupt priority scheme (represents internal state of kernel)
- Numbers 0-31; higher number has higher priority
- HAL maps hardware interrupts to IRQLs
- IRQL is attribute of interrupt resource (scheduling priority is attribute of thread)
- Lazy IRQL (don‘t mask lower interrupts on PIC until lower interrupt occurs)
- Try to avoid high IRQL

IRQLs (2)



Predefined IRQLs

- *High*: halting system or masking out all interrupts
- *Power fail*: power failure (not used)
- *Inter-processor interrupt*: request another processor to perform action
- *Clock*: clock interrupt (measure time and time slices)
- *Profile*: used when kernel profiling is enabled
- *Device*: the actual device interrupts
- *DPC/dispatch and APC*: software interrupts generated by kernel and device drivers
- *Passive*: normal thread scheduling and execution

IRQL Restrictions

- Only non paged memory can be accessed at IRQL DPC/dispatch or higher:
  - When page fault occurs, memory manager initiates disk I/O and waits for file system driver
  - Requires scheduler to perform context switch → violates rule that scheduler can‘t be invoked (scheduler works at IRQL *dispatch*)
  - System crashes with IRQL_NOT_LESS_OR_EQUAL

Interrupt Object

- When interrupt occurs a few assembly instructions are executed (*dispatch code*)
- Dispatch code is copied from template and stored in interrupt object
- Dispatch code calls interrupt dispatcher (passes pointer to interrupt object)
- Dispatcher uses object to locate registered Interrupt Service Routine(s) (ISR) and calls it (them)
- Interrupt object also contains associated IRQL
- Dispatch routine raises IRQL before calling ISR and lowers it afterwards

Interrupt Object (2)

- On multi-processor system exists one interrupt object for each CPU
- ISR is connected and disconnected to interrupt level via kernel functions (driver can turn ISR „on“ and „off“)
- Using interrupt object eliminates need to register ISR directly to hardware (HW independent)
- Interrupt object is used to synchronize execution of ISR
- Interrupt object allows to „daisy-chain“ (different) ISRs for a specific interrupt level (ISR has to acknowledge interrupt if it services it)

Software Interrupts

- Initiate thread dispatching
- Non-time critical interrupt processing
- Handling timer expiration
- Asynchronous execution of a procedure in the context of a particular thread
- Supporting asynchronous I/O operations

Thread Dispatching

- When thread can no longer execute (e.g. is terminated) or voluntarily enters wait state, kernel calls dispatcher directly
- But if scheduling is not appropriate at a given moment a DPC software interrupt is initiated
- E.g. *Clock* interrupt checks if current time quantum is expired and initiates scheduling DPC
- DPCs are used by device drivers to defer less time-critical activity
- Kernel raises IRQL to DPC level if shared kernel structures are accessed (scheduler has exclusive access to scheduling queues)

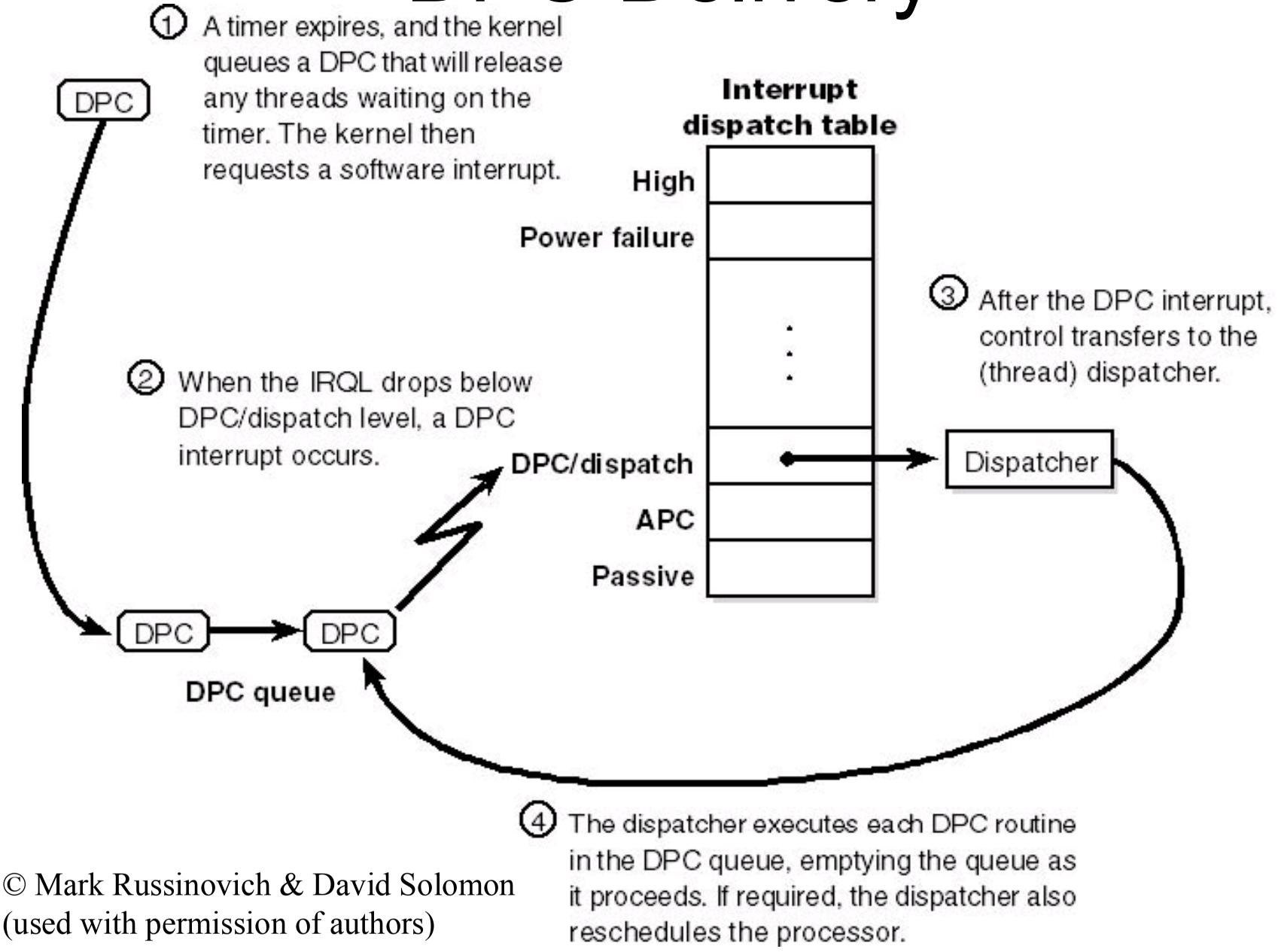
Deferred Procedure Call

- Represented by DPC object
- Objects contain address of DPC routine
- Stored in per-processor queues
- DPC can be queued with DPC priority (start, end of queue – see next slide)
- DPC can be aimed at particular processor

DPC Interrupt Generation Rules

DPC Priority	DPC Targeted at ISR's Processor	DPC Targeted at Another Processor
Low	DPC queue length exceeds maximum DPC queue length or DPC request rate is less than minimum DPC request rate	DPC queue length exceeds maximum DPC queue length or System is idle
Medium	Always	DPC queue length exceeds maximum DPC queue length or System is idle
High	Always	Always

DPC Delivery



Asynchronous Procedure Call

- APCs are queued to execute in context of particular user thread
- APC routine can acquire resources, wait on object handles, incur page faults, and call system services
- Described by APC object
- Objects in thread-specific APC queue
- APC executed when thread is scheduled
- „kernel“-mode APCs don‘t need permission of target thread to run in its address space (e.g. Used by POSIX to deliver signals)
- „user“-mode APCs do

APC (2)

- Asynchronous I/O uses APCs to deliver response
- Some Win32 API calls use „user“-mode APC for answer by specifying call-back routine
- Thread has to be in *alertable wait state* to accept APC
- Reaches this state by:
  - Waiting on object handle
  - Specify that wait is alertable
  - Directly testing for APC

Exception Dispatching

- On x86, all exceptions have predefined interrupt numbers and trap handlers in IDT
- Exceptions are serviced by exception dispatcher
- Dispatcher has to find an exception handler
- The kernel traps and handles some exception transparently to user (debug breakpoint)
- A few exceptions are allowed to filter back, untouched, to user mode (subsystem has to deal with exception)

Exception Dispatching (2)

- When exception occurs CPU transfers control to kernel trap handler
- Creates trap frame and exception record (reason and other information)
- If exception occurred in kernel mode, routine to locate frame-based exception handler called (unhandled kernel-mode exceptions are considered fatal)

User-Mode Exception

- In user mode, debugger breakpoints are often source → check for associated debugger
- Sends first chance notification to debugger
- If no debugger found or debugger does not handle exception, switched to user-mode and find frame-based exception handler

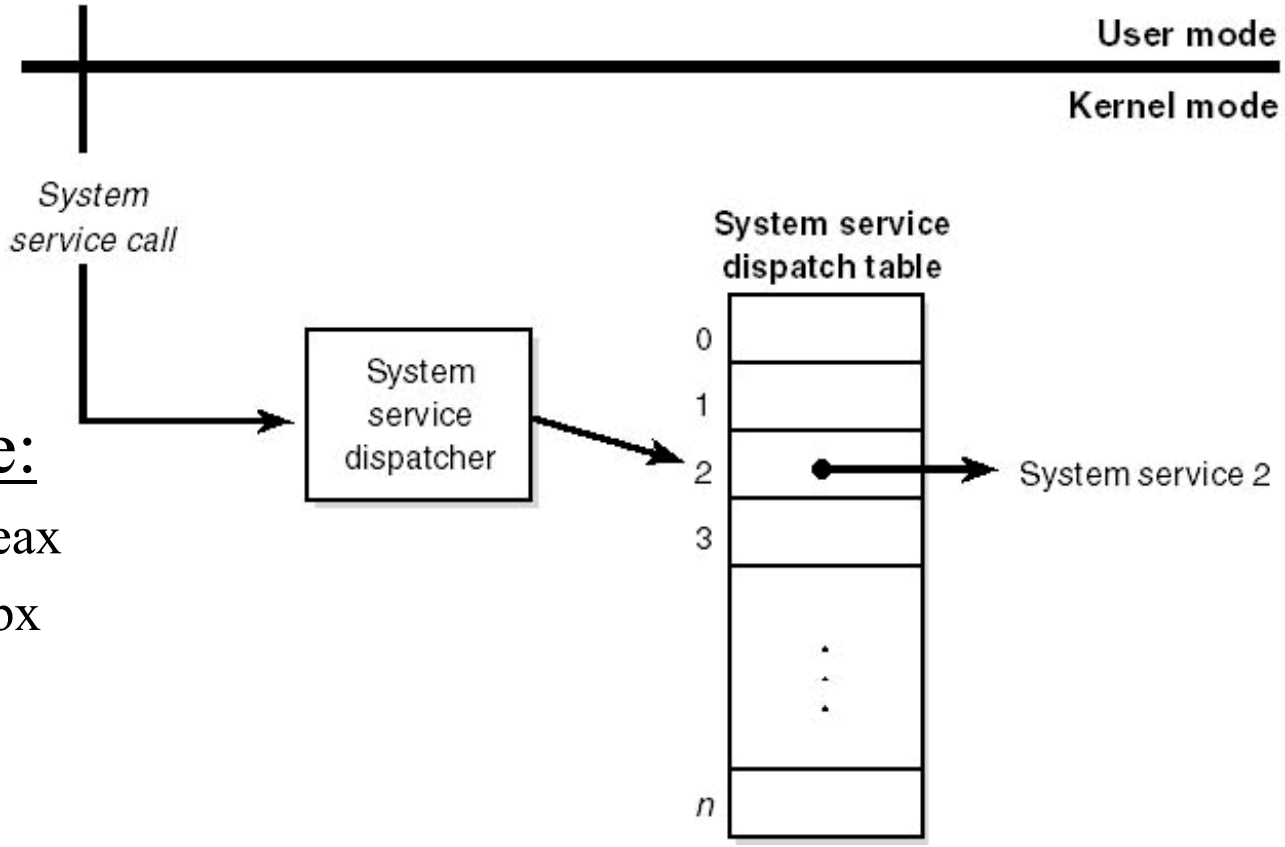
User-Mode Exception (2)

- If none found or exception not handled, switched to kernel-mode and second chance notification send to debugger
- All Win32 threads have exception handler at top of stack to process unhandled exceptions
- Default „debugger“ on Win2K is Dr. Watson (post-mortem tool – records state of app.)

Services

- System entry to use service is software interrupt (int 0x2E)
- System service dispatcher is called
- In register EAX is service number
- In register EBX is pointer to parameter list
- System service dispatcher:
  - Validates number of arguments and
  - Copies arguments from user mode stack to kernel-mode stack

Transition into kernel mode



NtWriteFile:

```
mov  $0x0E, %eax
mov  %esp, %ebx
int    $0x2E
ret   $0x2C
```

Transition into kernel mode (2)

- All validity checks are done after user to kernel transition
  - *KiSystemService* probes argument list, copies to kernel-mode stack and calls routine
  - Service-specific routine checks values, probes pointers
  - Once past that point, everything is „trusted“
- This is safe, because:
  - System service table is in kernel-protected memory
  - Kernel-mode routines are in kernel-protected memory
  - User mode can‘t supply the code to be run in kernel-mode; it can only select from predefined list
  - Arguments are copied to kernel-mode stack before validation → other threads can‘t corrupt the arguments