Operativsystem

Systemanrop

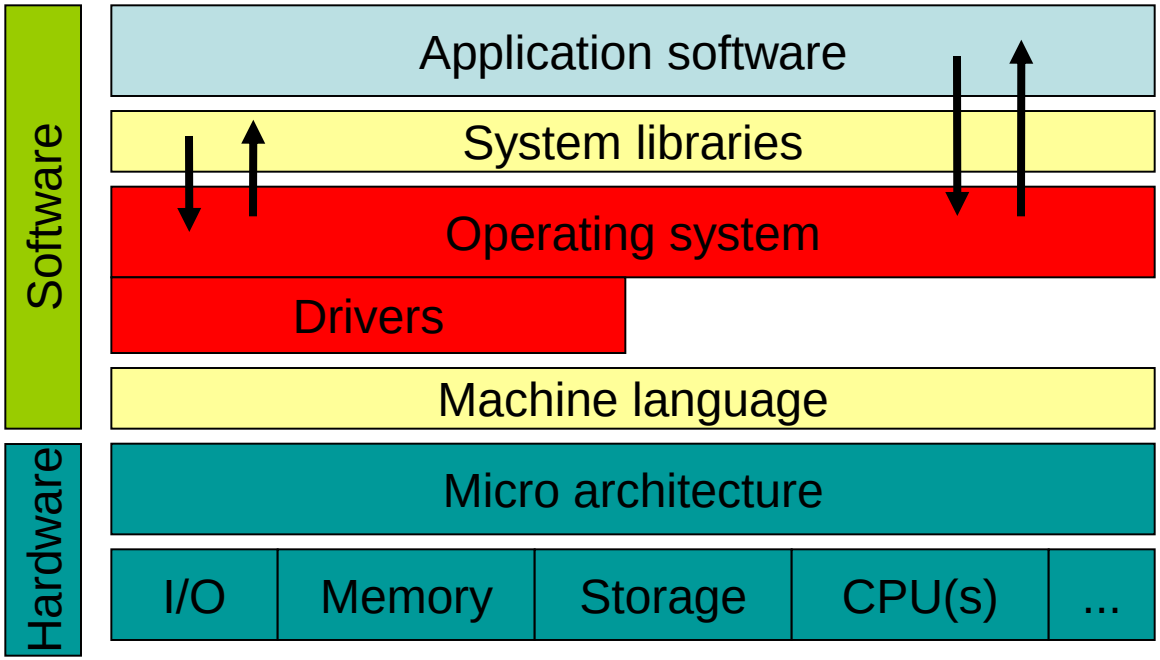
(1.6 i boken)

Systemanrop

- Gränssnittet mellan operativsystem och användarprogram sköts med systemanrop
- Systemanrop erbjuder service till de tjänster som OS sköter om, dvs.:
  - Generalisering av hårdvaran
  - Resursadministration

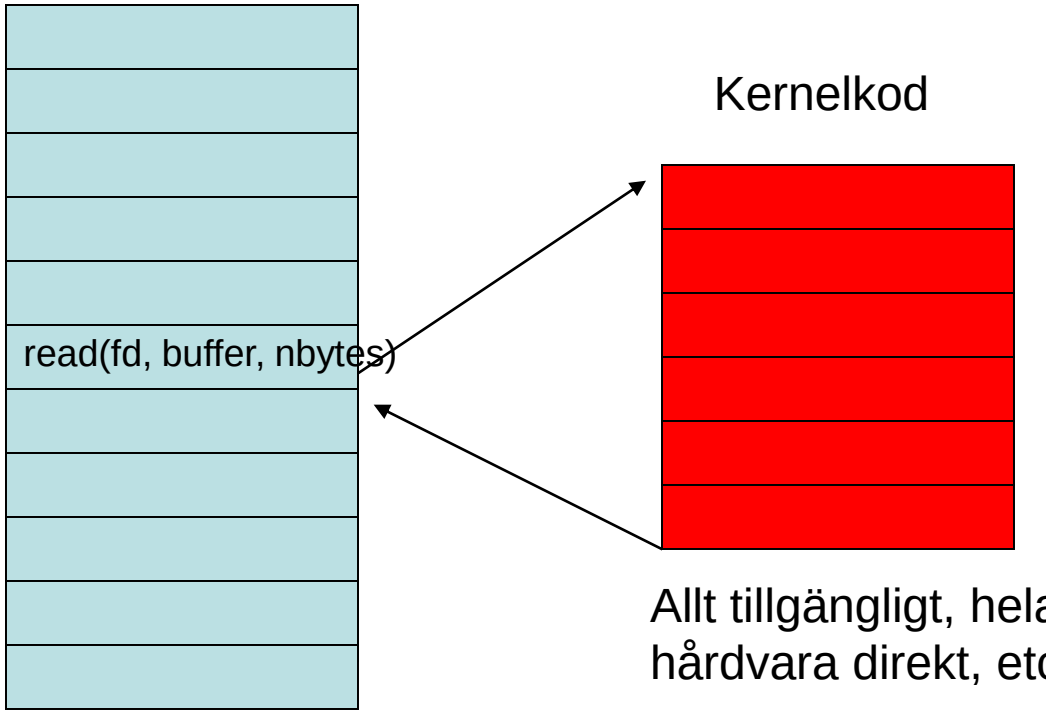
Systemanrop exemplifierar bra vad ett operativsystem egentligen handlar om

Den generella bilden



Systemanrop

Användarprogram



Allt tillgängligt, hela minnet, hårdvara direkt, etc....

Skyddat minne, får ej komma åt hårdvara

Systemanrop

- Generellt som vilket funktionsanrop som helst
  - “Manglas” dock till ett systemanrop, en speciell struktur

Unix: Posix:

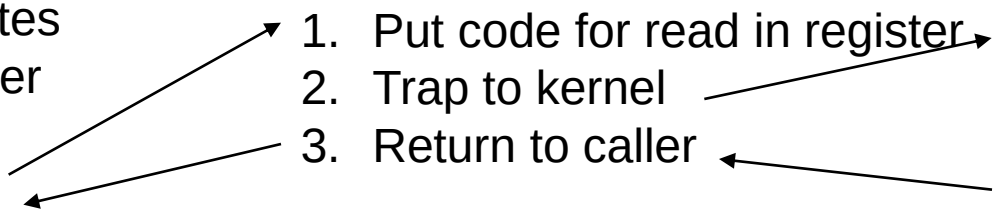
count = read(fd, buffer, nbytes);

Win32 API:
ReadFile(hFile, lpBuffer, nBytes, lpnBytesRead, lpOverLapped)
(But not mapped directly to syscall)

Application:
1. Push nbytes
2. Push buffer
3. Push fd
4. Call read

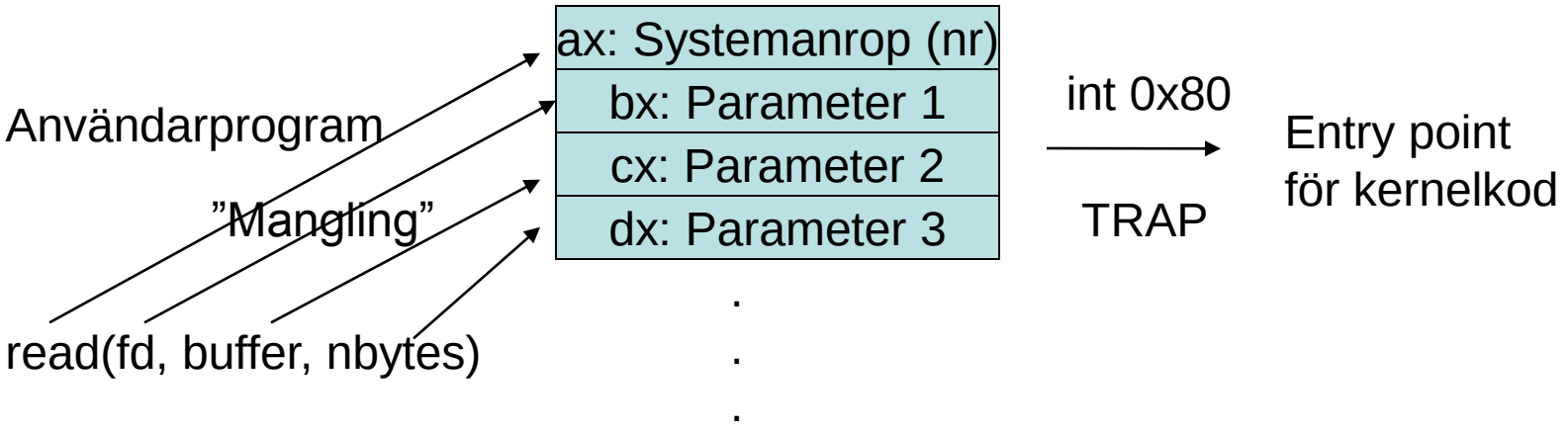
System library:
1. Put code for read in register
2. Trap to kernel
3. Return to caller

Kernel
1. Decode code
2. Check parameters
3. Sys call handler
4. Return



entry.S

Systemanrop



Dvs. ALLA FUNKTIONSANROP GÅR
GENOM SAMMA ”PORT” TILL KÄRNAN

Systemanrop

- Unix (Linux):
  - Anropsnummer i EAX
  - Interrupt 0x80
  - Parametrar i register

- Win32
  - Anropsnummer i EAX
  - Interrupt 0x2EH
  - Parametrar på stacken, pekare i EDX

WinNT syscall (i386)

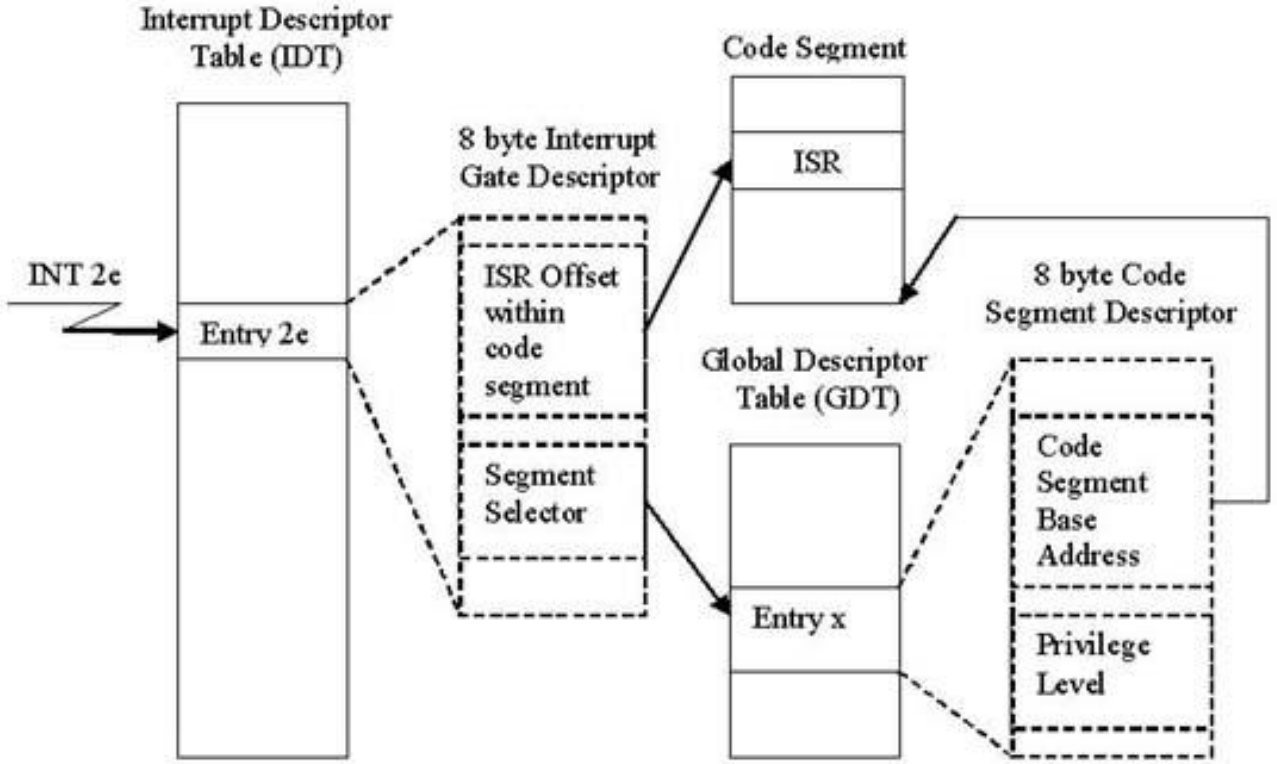


Figure 3. How the CPU finds the Interrupt Service Routine for software interrupt 2e.

Exampel: Lista systemanrop

- HelloWorld.c

```
[jbjorkqv@borken ~/tmp]$ more HelloWorld.c
#include <stdio.h>

int main() {
  printf("Hello world\n");
  exit(0);
}

[jbjorkqv@borken ~/tmp]$ gcc HelloWorld.c –o HelloWorld
[jbjorkqv@borken ~/tmp]$ strace ./HelloWorld
```

Exampel: Lista systemanrop (2)

```
$> strace ./a.out

execve("./a.out", ["./a.out"], [/* 24 vars */]) = 0
uname({sys="Linux", node="borken", ...}) = 0
brk(0)                                  = 0x804962c
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
    0x40016000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=82445, ...}) = 0
old_mmap(NULL, 82445, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40017000
close(3)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0 \304\1"..., 1024) = 1024
fstat64(3, {st_mode=S_IFREG|0755, st_size=5735106, ...}) = 0
old_mmap(NULL, 1267176, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x4002c000
mprotect(0x40158000, 38376, PROT_NONE)  = 0
old_mmap(0x40158000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x12b000) =
    0x40158000
old_mmap(0x4015e000, 13800, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
    -1, 0) = 0x4015e000
close(3)                                = 0
munmap(0x40017000, 82445)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
    0x40017000
write(1, "Hello world\n", 12Hello world
)           = 12
munmap(0x40017000, 4096)                = 0
_exit(0)
```

Typer av systemanrop

- Processhantering
- Minneshantering
- Filhantering
- Kataloghantering
- Allmänna systemsanrop

Linux 2.6.34 – 299 systemanrop

`/usr/include/asm/unistd.h`

`See also: man syscalls`

Win Vista – 360 system calls

`NtXXXXX – e.g. NtOpenFile()`

`http://www.metasploit.com/users/opcode/syscalls.html`

Systemanropslista (Linux)

```
% more /usr/include/asm/unistd.h

#define __NR_exit                  1
#define __NR_fork                  2
#define __NR_read                  3
#define __NR_write                 4
#define __NR_open                  5
#define __NR_close                 6
#define __NR_waitpid               7
#define __NR_creat                 8
#define __NR_link                  9
#define __NR_unlink               10
#define __NR_execve               11
#define __NR_chdir                12
#define __NR_time                 13
#define __NR_mknod                14
#define __NR_chmod                15
#define __NR_lchown               16
#define __NR_break                17
#define __NR_oldstat              18
#define __NR_lseek                19
#define __NR_getpid               20
#define __NR_mount                21
.
.
.
```

Ex: Processhantering

pid = fork()

Skapa en process

pid = waitpid(pid, &statloc, options)

Vänta på process

s = execve(name, argv, environp)

Byt ut ”koden” för en process

exit(status)

Avsluta processen

Typisk processhantering

- Kommando-tolk:

```
while (1) {
    type_prompt();
    read_command(command, parameters);
    if (fork() != 0) {
        /* parent */
        waitpid(-1, &status, 0); /* wait for child */
    } else {
        /* Child */
        execve(command, parameters, 0);
    }

}
```

Ex: Minneshantering

*brk(void *end_data_segment)*

Sätt slutet på processens datasegment

sbrk(increment)

Inkrementera processens datasegment

*getrlimit(int resource, struct rlimit *rlim)*

*setrlimit(int resource, struct rlimit *rlim)*

Sätt resursbegränsning för process (kodstorlek, adressrymd, antal öppna filer)

Ex: Filhantering

fd = open(filename, flags)

Öppna en fil

close(fd)

Stäng fil

n = read(fd, buffer, nbytes)

Läs från fil

n = write(fd, buffer, nbytes)

Skriv till fil

position = lseek(fd, offset, whence)

Gå til position i filen

s = stat(name, &buf)

Hämta status för filen

Ex: Kataloghantering

mkdir(name, mode)

Skapa katalog

rmdir(name)

Radera katalog

link(name1, name2)

Skapa länk från name1 till name 2

unlink(name)

Radera katalog

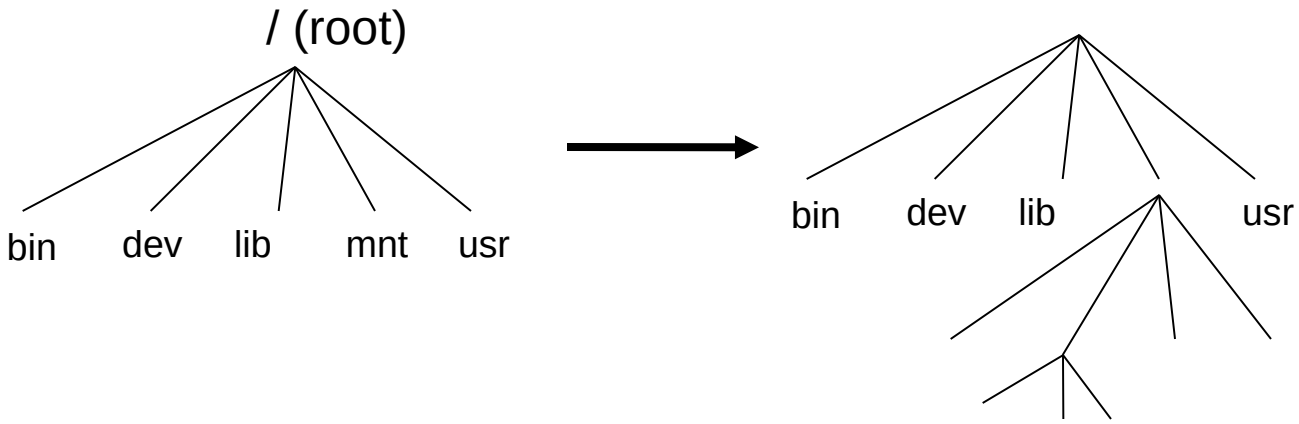
mount(special, name, flag)

”Mounta” ett filsystem

umount(special)

Kataloghantering

- mount (”dev/fd0”, ”/mnt”, 0)



Allmänna

s = chdir(dirname)

Ändra aktiv katalog

s = chmod(name, mode)

Ändra skyddsbitar för fil

s= kill(pid, signal)

Sänd signal till process

seconds = time(&seconds)

Ge tid sedan 1. Jan 1970 (UNIX)

Win32 och systemanrop

- I Unix många gånger en direkt mappning från biblioteksfunktion till systemanrop
  - t.ex. funktionen read() mappas direkt till systemanropet read()
- Win32 däremot har ett API, som effektivt ”gömmer” alla systemanrop
- Gör det möjligt att ändra systemanrop bara bibliotek (dvs Win32 API) uppdateras att motsvara det nya systemanropet

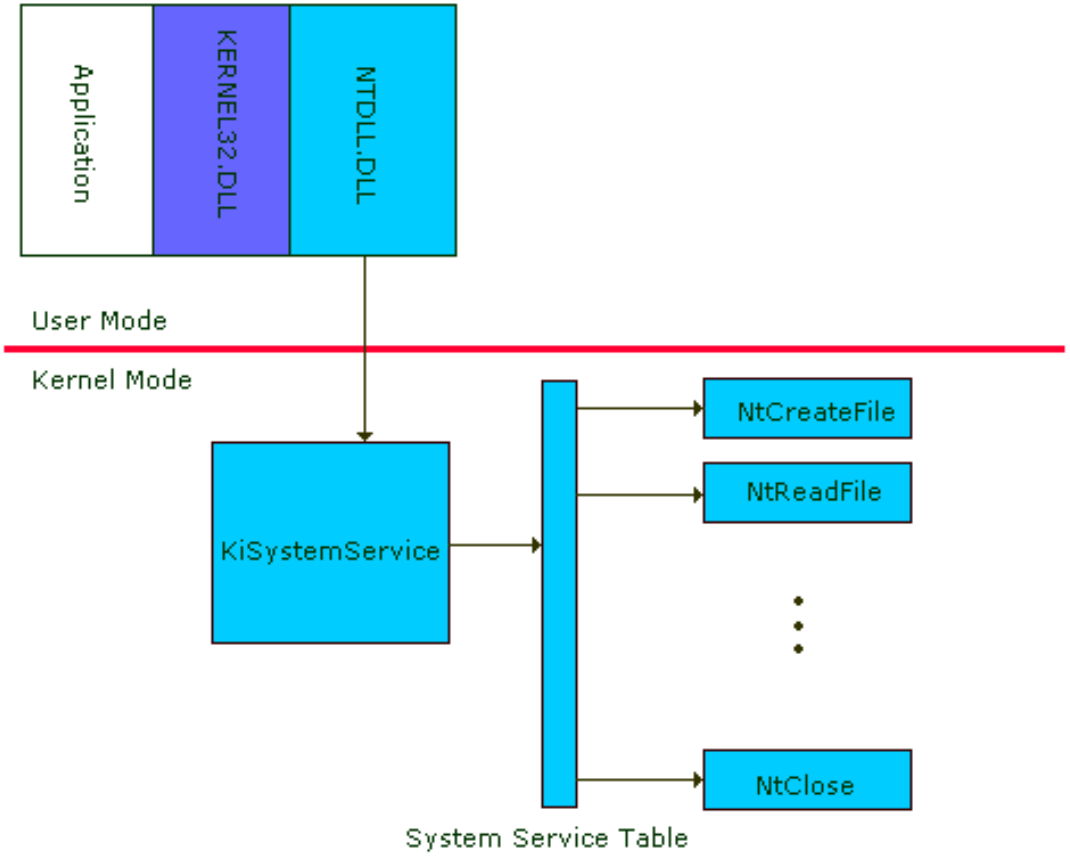
NT Native API

- Win NT gömmer nästan alla sina systemanrop, orsak: Struktur på NT: Modifierad microkernel
- NT implementerar OS-delsystem i ”user mode”
  - CSRSS.EXE – Client/Server runtime system – serverprocess
  - Dynamiska bibliotek använda av klienter (dvs användarprogram)
    - USER32.DLL – Windows-and messaging
    - GDI32.DLL – Grafikfunktioner
    - KERNEL32.DLL – I/O, processhantering, minneshantering, synkronisering

Win32 API-anrop

- Då ett dynamisk bibliotek anropas sker någondera av följande
  - returnerar direkt (eftera att endast intern gjort operationer)
  - meddelande skickas till Win32-serverprocessen för att be om hjälp
  - kallar på Native API för att utföra operationen

Win32 Native API-struktur



Native API – parameter-check

- KiSystemService borde granska parametrarna som ges till systemanrop noggrant
- Mark Russinovich rapporterar:
  (www.sysinternals.com , nuförtiden del av microsoft.com)
  - NT4: ca 50 Native API-funktioner som inte checkar parametrar -> blue screen

Native API - exempel

-
- Processes and Threads
- These functions control processes and threads. Many have direct Win32 equivalents.
- NtAlertResumeThread Resumes a thread.
- NtAlertThread Sends an alert to a thread.
- NtTestAlert Tests for whether a thread has a pending alert.
- NtCreateProcess CreateProcess Creates a new process.
- NtCreateThread CreateThread Creates a new thread.
- NtCurrentTeb Returns a pointer to a thread's environment block.
- NtDelayExecution Sleep, SleepEx Pauses a thread for a specified time.
- NtGetContextThread GetThreadContext Retrieves the hardware context (registers) of a thread.
- NtSetContextThread SetThreadContext Sets the hardware context (registers) of a thread.
- NtOpenProcess OpenProcess Opens a handle to a specified process. DDK
- NtOpenThread OpenThread Opens a handle to a specified thread.
- NtQueryInformationProcess GetProcessTimes, GetProcessVersion, GetProcessWorkingSetSize, GetProcessPriorityBoost, GetProcessAffinityMask, GetPriorityClass, GetProcessShutdownParameters Obtains information about a process' attributes. DDK
- NtQueryInformationThread GetThreadTimes, GetThreadPriority, GetThreadPriorityBoost Obtains information about a thread's attributes. DDK
- NtQueueApcThread QueueUserApc Introduced in NT 4.0. Queues an Asynchornous Procedure Call to a thread.
- NtResumeThread ResumeThread Wakes up a suspended thread.
- NtSetInformationProcess SetProcessAffinityMask, SetPriorityClass, SetProcessPriorityBoost, SetProcessShutdownParameters, SetProcessWorkingSetSize Sets a process' attributes. DDK
- NtSetInformationThread SetThreadAffinityMask, SetThreadIdealProcessor, SetThreadPriority, SetThreadPriorityBoost Sets a thread's attributes. DDK
- NtSetLowWaitHighThread NT 4.0 only (not in Win2K).
- NtSetHighWaitLowThread NT 4.0 only (not in Win2K).
- NtSuspendThread SuspendThread Suspends a thread's execution.
- NtTerminateProcess TerminateProcess Deletes a process.
- NtTerminateThread TerminateThread Deletes a thread.
- NtYieldExecution SwitchToThread Introduced in NT 4.0. Causes thread to give up CPU.
-

Avbrott

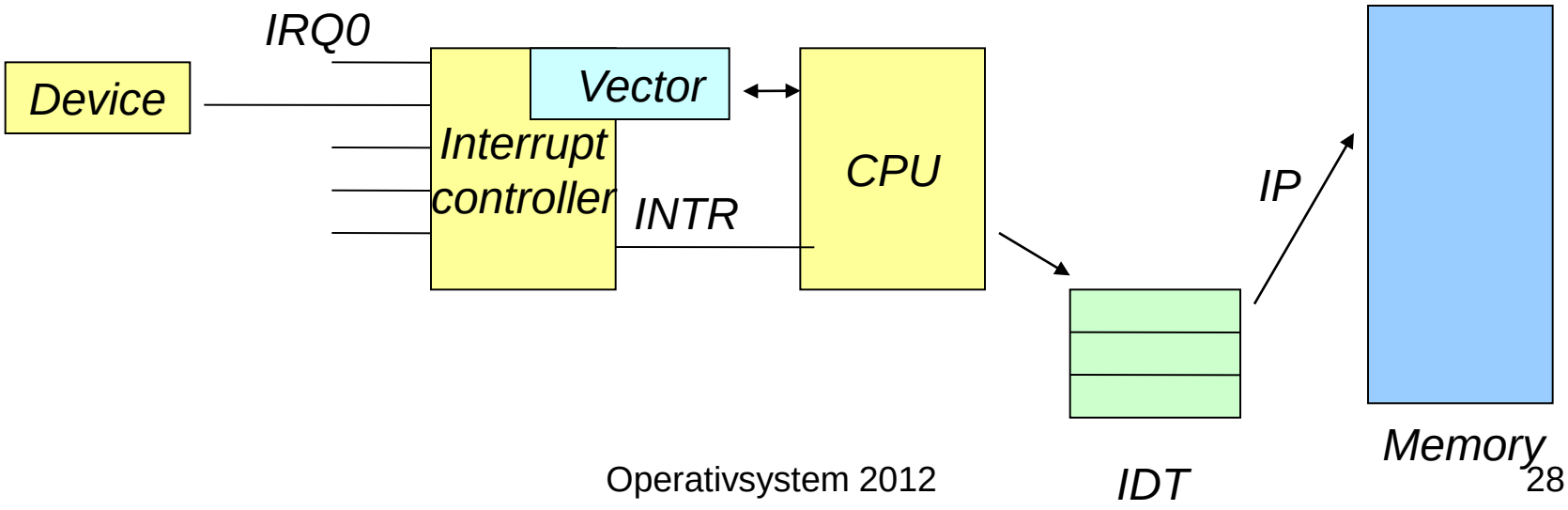
- Generellt: Avbrott tvingar CPU:n att avbryta den aktiva exekveringsstigen och utföra kod som specificeras av en avbrottsvektor
- Källor till avbrott
  - Enheter (I/O)
  - Klocka
- Avbrott kan vara
  - Maskable
  - Non-maskable (NMI)

IRQ och avbrott

- IRQ = Interrupt request
- När en enhet förorsakar en IRQ, händer följande
  - Avbrottskontrolleraren (Interrupt Controller), eller Programmable Interrupt Controller (PIC)
    - Konverterar IRQ till motsvarande vektor
    - Sparar vektorn i avbrottskontrolleraren
    - Skapar en signal på CPU:s INTR-pin – detta är ett avbrott
    - Väntar tills CPU:n meddelar att den tagit emot avbrottet genom att skriva till avbrottskontrolleraren

Avbrottdeskriptortabell

- I avbrottdeskriptortabellen (IDT) associeras varje interrupt eller exception med adressen till motsvarande hanterare



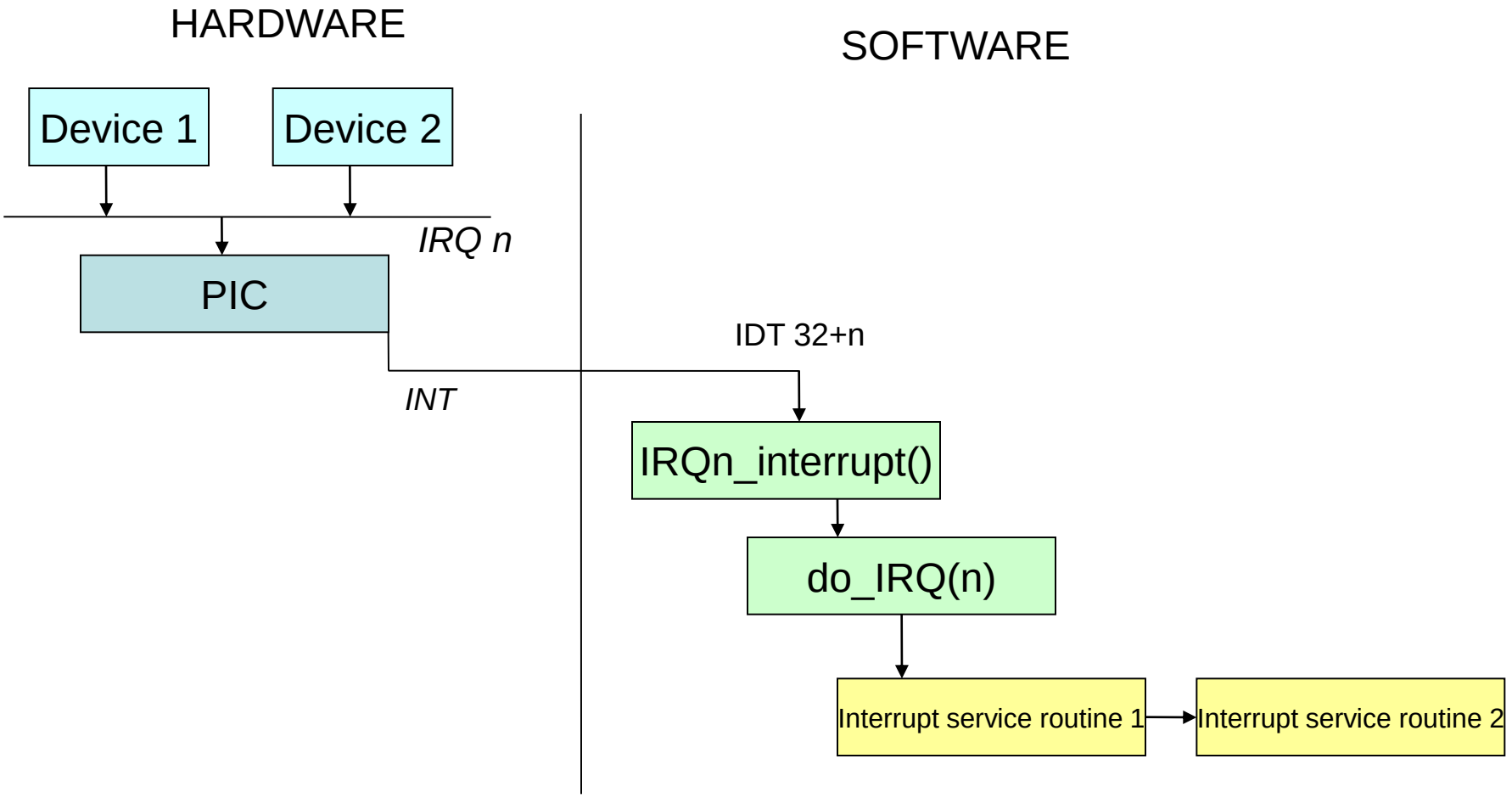
Delade IRQ

- IRQ-linjer är begränsade (I PC-arkitektur 8 per PIC, normalt 2 PIC-> 16 IRQ)
- Enheter måste dela avbrottslinjer
  - Avbrottshanteraren måste kunna klara av flera enheter
    - En lista av servicerutiner
  - All hårdvara stöds dock inte av detta

Avbrottshanterare

1. Spara avbrottets nummer samt CPU:s register
2. Meddela att avbrottet är emottaget
3. Exekvera avbrottsrutin för detta avbrott
4. Återvänd från avbrott

I/O-avbrottshantering



Responstid i OS

- Då avbrottsrutinen exekveras gör systemet INGET ANNAT
  - Är rutinen lång, blir responstiderna långa
- Gör så litet som möjligt i avbrottsrutinen
  - Det icke-kritiska kan göras senare, med hjälp av system som OS erbjuder (softirqs, tasklets and bottom halves)
  - Avbrott avstängda endast under kritiska delar

SoftIRQs, tasklets och bottom halves

- Several things done in an interrupt handler may be not critical
- Flere operationer i en avbrotts-hanterare kan vara icke-kritiska
- Uppdelning enligt prioritet
  - Kritiska – Avbrott avstängda
    - Kvittera avbrottet, uppdatering av datastrukturer som används av processorn och enheten
  - Icke-kritiska – Avbrott aktiverade
    - Uppdatering av datastrukturer
  - Icke-kritiska, som kan senareläggas – Softirq, taslkets, bottom halves
    - Kopiering av buffrar

Senarelagda funktioner

- Softirq
  - Softirqs av samma typ kan exekveras samtidigt på flere CPUn, men en softirq kan inte avbrytas av en annan softirq på samma CPU
- Tasklet (normala modellen för senarelagda funktioner)
  - Tasklets av olika typ kan exekveras samtidigt på flere CPU, men tasklets av samma typ kan ej avbryta varandra
- Bottom half
  - Bottom halfs är globalt serialiserade (endast en åt gången kan exekveras i systemet)

Exceptioner

- En exeption är ett ”synkront” avbrott
  - Produceras av CPU efter utförande av en instruktion
- 80x86-processorfamiljens definieras ca 20 exeptioner
  - Divide error, debug, NMI, breakpoint, overflow, bounds check, invalid op-code, device not available, double fault, segment not present, stack segment, general protection, page fault, floating-point error, alignment check, machine check, SIMD floating point
- Kan normalt hanteras som ett vanligt avbrott

Typer av exceptioner

- Faults (felsituationer)
  - Kan normalt repareras, efter reparationen fortsätter programmet normal
    - Instruktionen som förorskakade felsituationen utförs på nytt
- Traps (fälla)
  - Rapporteras efter instruktionen som förorsakade “trap” har utförts, använda normalt vid debuggning
    - Instruktionen efter trap-instruktionen utförs till näst
- Aborts
  - Allvarliga fel, t.ex. hårdvarufel
- Programmerade avbrott, systemanrop
  - Normalt bruk: Systemanrop, debuggning av mjukvara