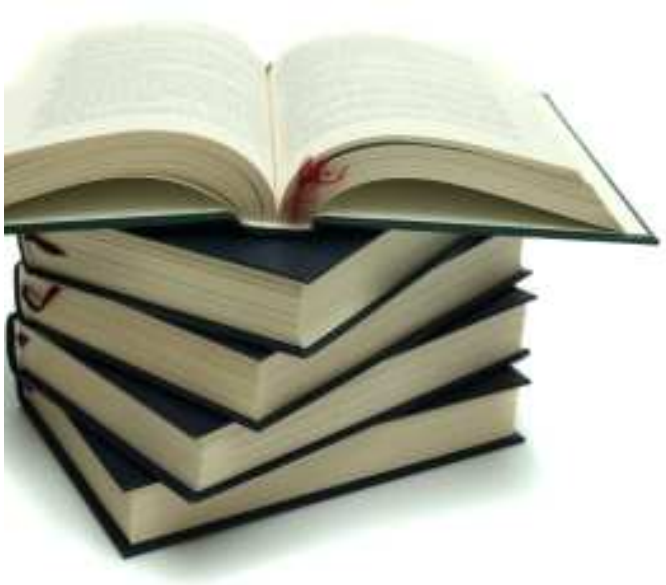
SO2
kernel crunch

Cursul 6



6
Întreruperi

2 aprilie 2009

- Scrieți în pseudo C/asamblare o funcție ce realizează o schimbare de context între două thread-uri userspace, presupunând că aveti access direct la toți regiştrii.
- Dupa fork(), este partajată tabela de file descriptori între procesul copil si părinte? Dar fişierele?
- Explicați condiția de cursă din secvența de mai jos:

```
sem_down()
{
if (count == 0)
   sleep_on(wq);
   ...
}
```

```
sem_up()
{
...
if (count == 1)
   wake_up(wq);
}
```

- Întreruperi şi excepții (x86)
- Întreruperi şi excepții în Linux
- Întreruperi şi excepții în Windows
- Acțiuni amânabile

Bibliografie

- UTLK: capitolul 10
- LKD: capitolul 5
- WI: capitolul 3 (System Service Dispatching)

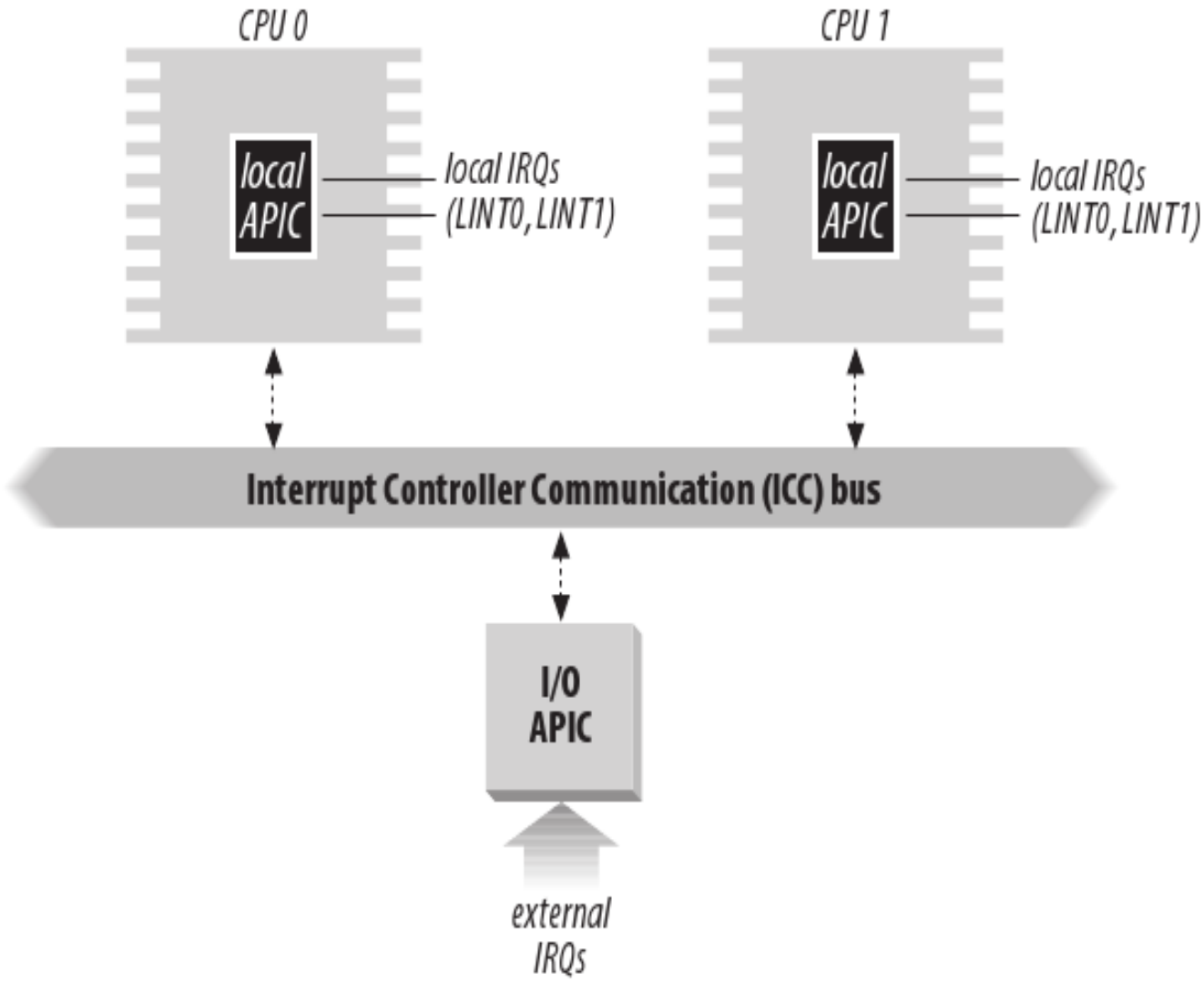
Întreruperi

- Un eveniment ce altearează fluxul normal de execuție
- Întreruperi sincrone (excepții) – întreruperea se generează ca urmare a execuției unei instrucțiuni
- Întreruperi asincrone – întreruperea se generează ca urmare a apariției unui eveniment extern procesorului
- Întreruperi mascabile - tratarea acestora poate fi oprită/pornită la cerere
  - Exemple: întreruperile generate de placa de rețea, disc, ceas, etc.
- Întreruperi nemascabile – întreruperea va fi tratată
  - Exemple: defecțiuni hardware, watchdog

Excepții

- Fault-uri
  - se salvează eip-ul instrucțiunii care a generat fault-ul
  - se poate corecta condiția care a determinat fault-ul şi relua
  - exemplu: page fault
- Trap-uri
  - se salvează eip-ul instrucțiunii următoare
  - nu se poate corecta condiția care a determinat trap-ul (şi nici nu este nevoie)
  - exemplu: debug, întreruperi software

Hardware


CPU 0
local APIC
local IRQs (LINT0, LINT1)
CPU 1
local APIC
local IRQs (LINT0, LINT1)
Interrupt Controller Communication (ICC) bus
I/O APIC
external IRQs

Activarea / dezactivarea întreruperilor

- Doar întreruperile mascabile sunt afectate
- Se foloseşte flagul IF din registrul EFLAGS pentru a ignora / trata întreruperile
- Imbricarea trebuie tratată de către sistemul de operare
- Instrucțiuni
  - CLI – Clear Interrupts
  - STI – Set Interrupts

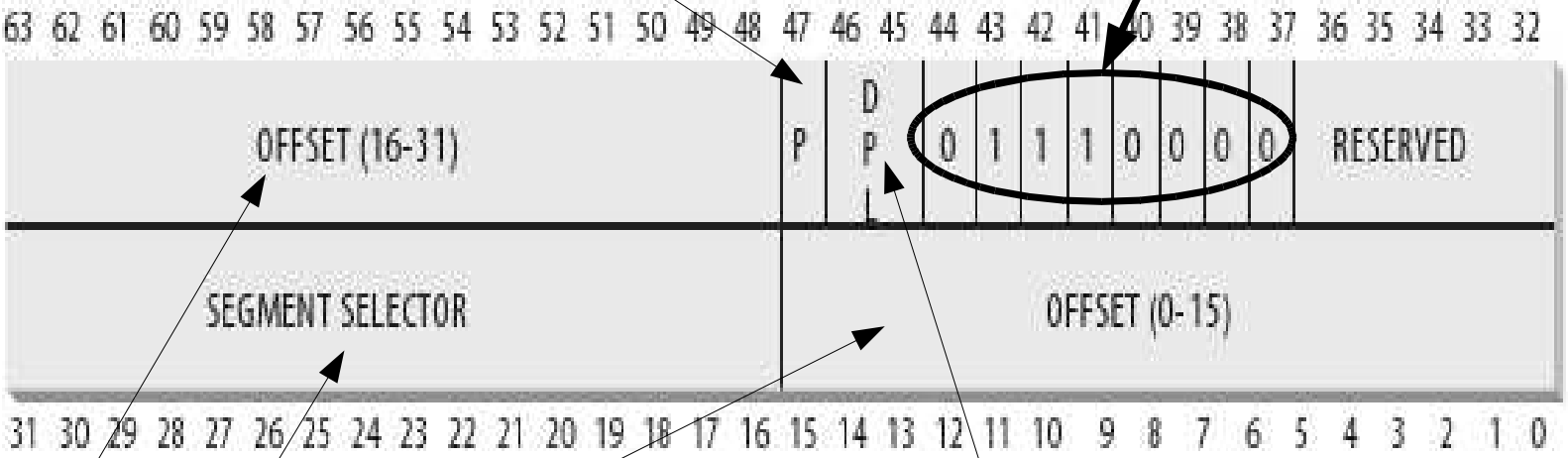
IDT

- Fiecare întrerupere/excepție are asociată un index
  - valori între 0 si 255
  - indexează Interrupt Descriptor Table
- Interrupt Descriptor Table
  - un vector (256 intrări) de descriptori
  - baza vectorului este indicată de valoarea registrului **idtr**
  - specifică modul de tratare al unei întreruperi/excepții

Descriptorul de întreruperi

Tipul descriptorului: Interrupt/Trap, TSS

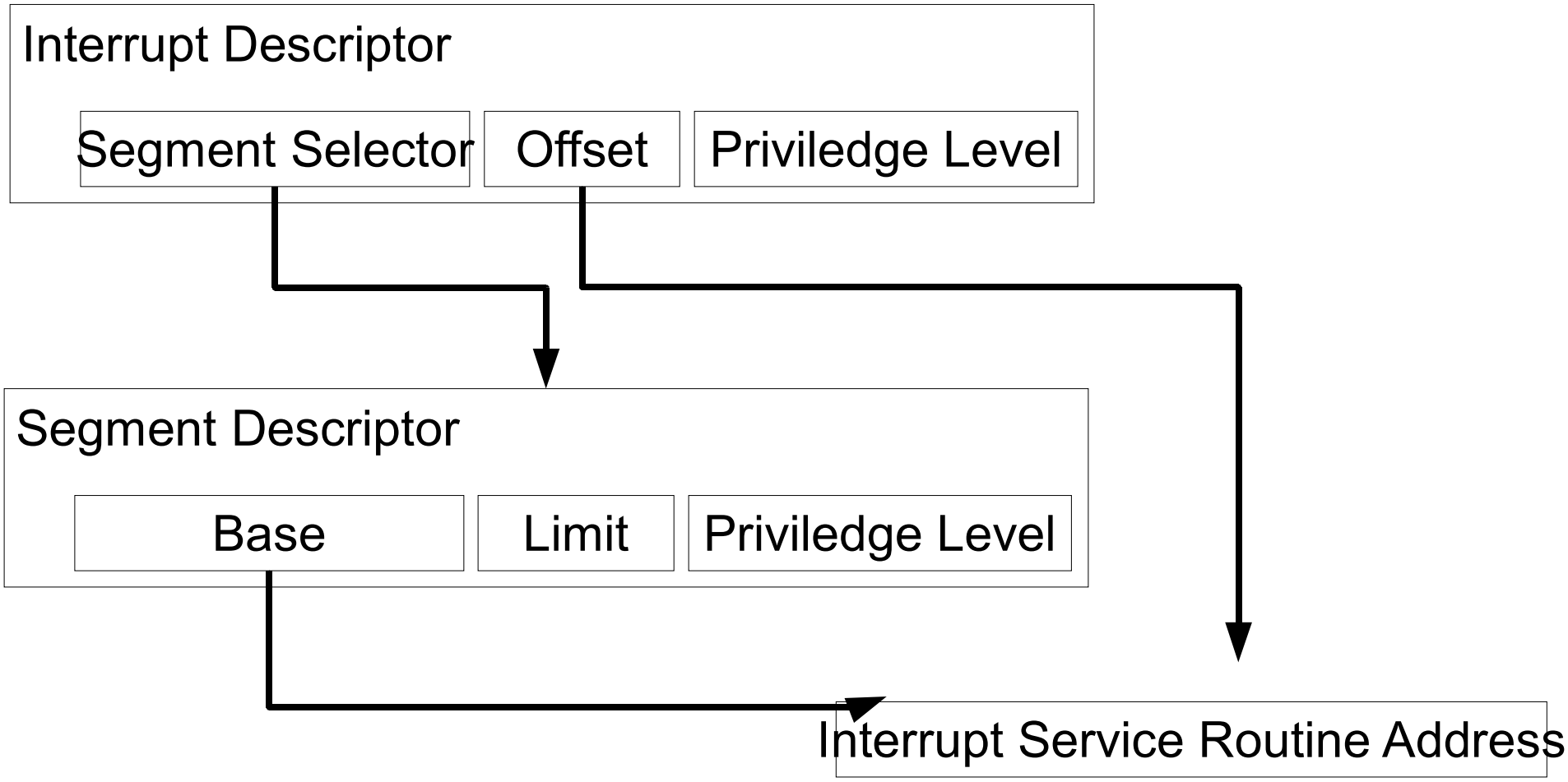
Present/Absent



Adresa rutinei de tratare

Nivelul de privilegiu necesar pentru a genera întreruperea: 3 pentru întreruperi software, 0 pentru alte excepții şi întreruperi hardware

Adresa rutinei de tratare a a întreruperii



Tipuri de descriptori

- Interrupt Gate Descriptor
  - Este folosit pentru întreruperi hardware
  - se setează IF pe 0
- Trap Gate Descriptor
  - Este folosit pentru excepții
  - IF nu este modificat
- Task Gate Descriptor
  - Proiectat pentru a face schimbarea de context, dar nefolosit în practică

Preemptarea întreruperilor

- Se face doar de către o altă întrerupere
- Procesele nu pot preempta întreruperile
- Din această cauză întreruperile trebuie să fie scurte, pentru a nu duce la process starvation

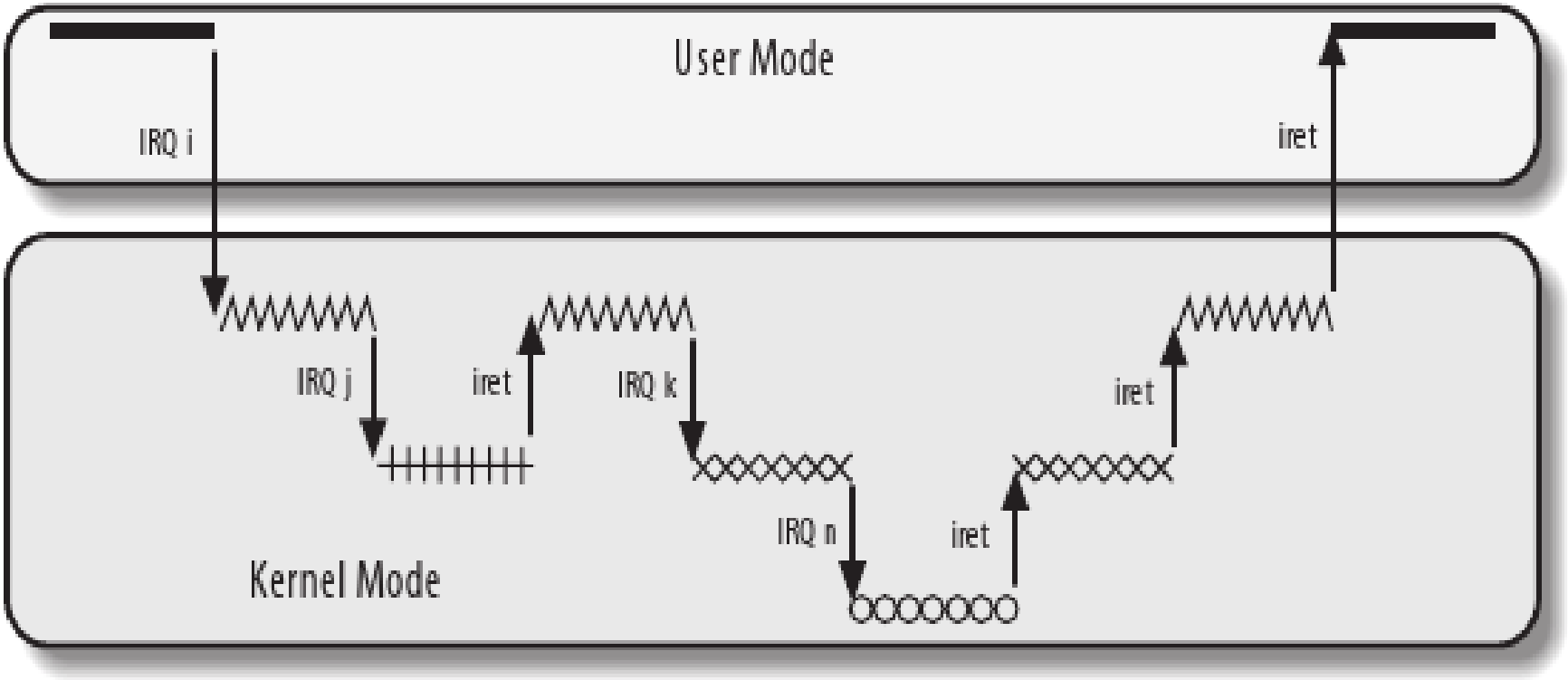
La generarea unei întreruperi...

- Se verifică nivelul curent de privilegiul cu DPL (atât în segment descriptor cât şi în interrupt descriptor)
- Dacă are loc o schimbare de nivel de privilegiu
  - Se schimbă stiva la cea asociată cu noul nivel de privilegiu (new ss, esp)
  - Se salvează pe noua stivă informații despre cea veche (old ss, esp)
- Dacă a fost generat un fault: se încarcă în **cs** şi **eip** adresa instrucțiunii care a generat fault-ul
- Daca avem un abort: se salvează codul erorii hardware pe stivă
- Se salvează **eflags**, **cs**, **eip** pe stivă
- Se execută rutina de tratare (**cs**←selector, **eip**←offset)cu nivelul de privilegiu dat de descriptorul de segment

La ieşirea din întrerupere

- Dacă am avut un abort: scoatem de pe stivă codul de eroare
- Se rulează instructiunea iret care
  - încarca **cs**, **eip** şi **eflags** cu valorile salvate pe stivă
  - dacă a avut loc o schimbare de privilegiu pune la loc stiva veche
  - revine la nivelul de privilegiu inițial

Întreruperi imbricate



- Avantaj: Procesarea unei întreruperi nu blochează device-ul sau controlerul de întreruperi

Imbricare întreruperi/excepții

- Excepțiile pot fi preemptate de întreruperi
  - Exemplu: apel de sistem, page fault, întrerupere
- În general, nu există necesitatea preemptării unei întreruperi de o excepție
  - Când se întâmplă acest lucru avem un kernel bug

Context întrerupere

- Codul ce rulează în urma declanşării unei întreruperi (nu şi excepții)
- Pentru că întreruperile sunt asincrone, codul ce se execută în context întrerupere nu are un context proces (bine definit)
- În context întrerupere NU se poate face sleep, apela schedule(), sau accesa memorie utilizator

SMP

- Controllerul de întreruperi distribuie întreruperile pe mai multe procesoare
  - Pentru througput maxim pentru ca întreruperea să fie ACKed cât mai repede
- Procesoarele comunică între ele prin IPI: Inter Processor Interrupts

Tratarea întreruperilor: faza 1

- Acțiuni critice
- Sunt executate cu întreruperile mascate, deci trebuie executate rapid
- Sunt executate înainte de rularea rutinei de tratare instalate de device driver
- Exemplu de astfel de acțiuni: ack PIC, reprogramare PIC

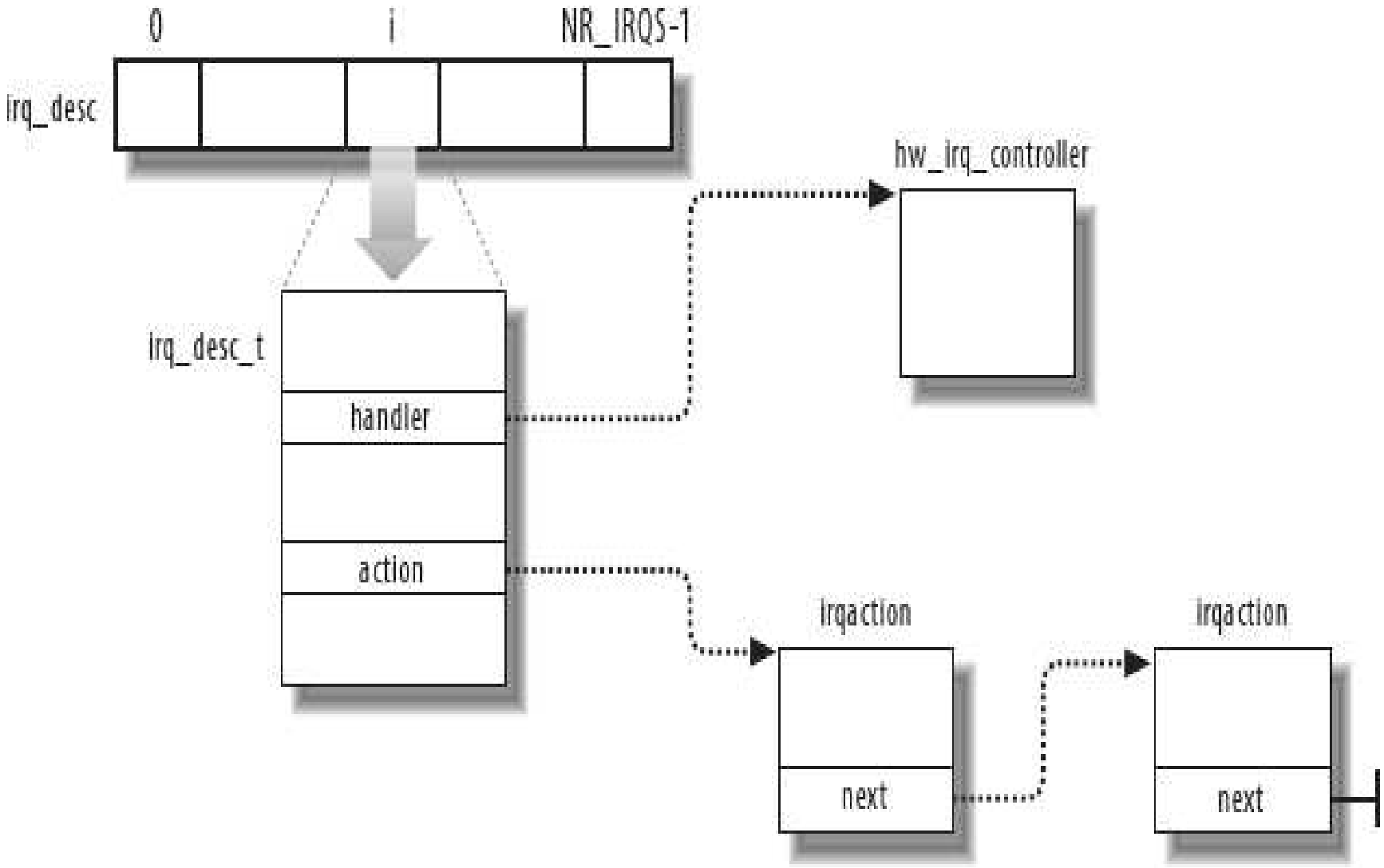
Tratarea întreruperilor: faza 2

- Acțiuni imediate
- Rularea rutinei de tratare instalate de device driver
- În general rutinele de tratare rulează cu întreruperea curentă mascată, dar cu întreruperile activate

Tratarea întreruperilor: faza 3

- Acțiuni amânabile
- acțiuni necritice ce pot fi executate mai târziu
- folosirea acțiunilor amânabile simplifică lucrul cu întreruperile
  - întreruperile nu trebuie să fie re-entrante
  - nu există riscul de stack overflow
  - se reduce probabilitatea pierderii de întreruperi

Tratarea întreruperilor în Linux



Tratarea întreruperilor în Linux (2)

- Nucleul interceptează toate întreruperile cu un handler generic
  - trimite ACK controlerului de întreruperi
  - rulează rutinele de întreruperi instalate de device drivere
- Implementează suportul pentru diverse mecanisme:
  - întreruperi partajabile
  - întreruperi întreruptibile
  - suport software pentru generarea de numere aleatoare
- Drivere pentru controllere de întreruperi:
  - Operații: startup, shutdown, enable, disable, ack, end

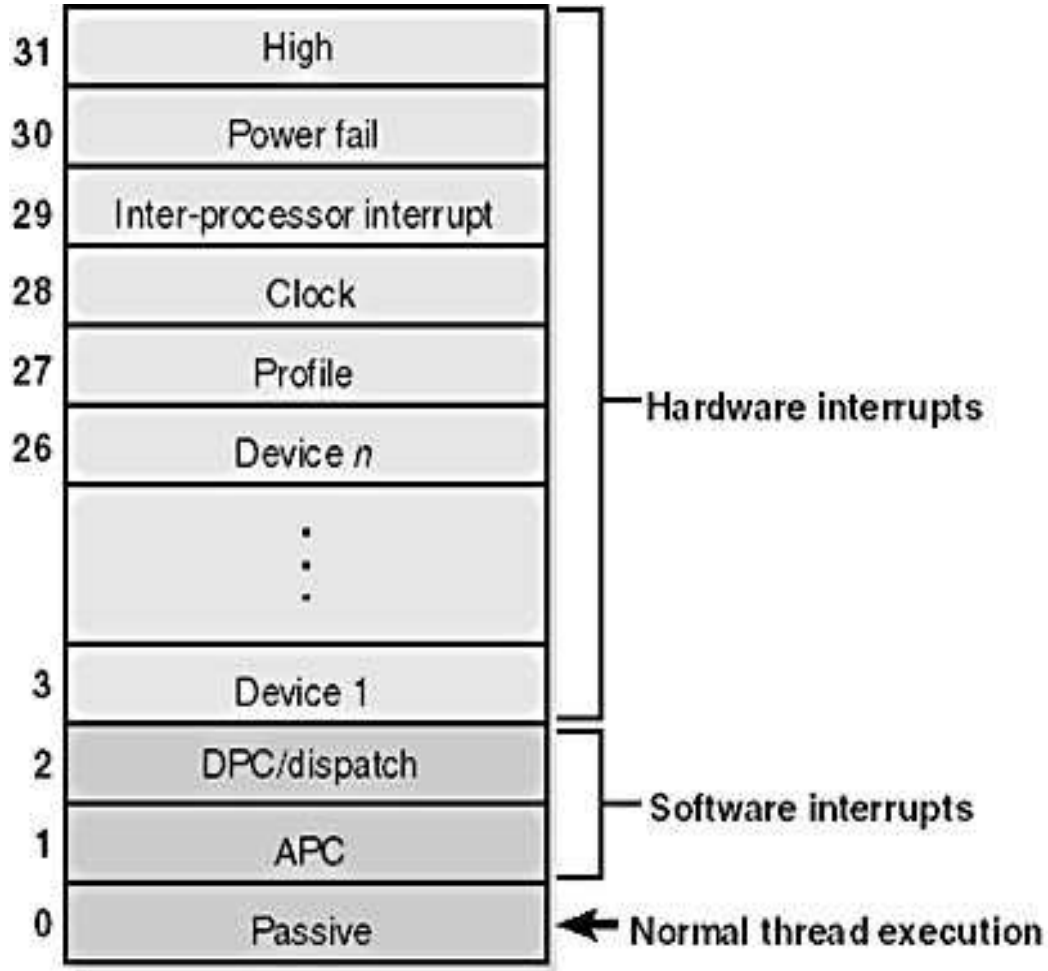
Tratarea întreruperilor în WNT

- WNT foloseşte un sistem de întreruperi cu priorităţi explicite (vizibile programatorului)
- Există un handler generic care tratează acțiunile critice
- Pentru instalarea handlerelor device driverele folosesc obiecte întrerupere
  - conectarea unui obiect întrerupere
  - deconectarea unui obiect întrerupere

IRQL

- Interrupt Request Levels
  - implementare software
  - fiecare întrerupere/excepție are o prioritate
  - întreruperile/excepțiile cu prioritate mai mare pot preempta întreruperi/execpțiile cu prioritate mai mică
- Lazy IRQL
  - pentru a ridica IRQL trebuie mascate anumite întreruperi
  - Lazy IRQL memorează nivelul (într-o variabilă) şi doar când apare o întrerupere/excepție se setează maschează întreruperile/excepțiile

IRQL (2)


31
High
30
Power fail
29
Inter-processor interrupt
28
Clock
27
Profile
26
Device n
3
Device 1
2
DPC/dispatch
1
APC
0
Passive
Hardware interrupts
Software interrupts
Normal thread execution

Acțiuni amânabile în Linux

- Sunt implementate cu ajutorul funcțiilor amânabile (deferrable functions)
- SoftIRQ (întreruperi software)
  - nu pot fi alocate dinamic
  - Aceeaşi întrerupere software poate rula concurent pe mai multe procesoare
- Tasklet
  - pot fi alocate dinamic
  - acelaşi tasklet nu poate rula concurent pe mai multe procesoare dar tasklet-uri diferite pot rula concurent

Întreruperi software

- Inițializarea: open_softirq()
  - o face utilizatorul:
- Activarea: raise_softirq()
  - o face utilizatorul în secțiunea imediată de tratare a întreruperii
- Execuția: do_softirq()
  - când se termină de tratat orice întrerupere
  - când rulează kernel thread-ul ksoftirqd
  - în cazuri speciale (susbsistemul de networking)

Tipuri de întreruperi software

- HI_SOFTIRQS
- TIMER_SOFTIRQ
- NET_TX_SOFTIRQ
- NET_RX_SOFTIRQ
- BLOCK_SOFTIRQ
- TASKLET_SOFTIRQ
- HRTIMER_SOFTIRQ

ksoftirqd

- Un thread kernel cu prioritate minimă ce execută unele întreruperi software
- O întrerupere software este planificată spre a rula în ksoftirqd dacă se reactivează singură
- Soluți de compromis între timp de răspuns bun atât pentru întreruperile software cât şi pentru procesele şi thread-urile din userspace

Taskleți

- Implementați peste întreruperile software
  - pot fi folosite două priorități: HI_SOFTIRQ şi TASKLET_SOFTIRQ
- Inițializarea: tasklet_init()
- Activarea: tasklet_schedule(), tasklet_hi_schedule()
- Mascarea: tasklet_disable(), tasklet_enable()

Acțiuni amânabile în WNT

- Deferred Procedure Call (DPC)
  - Similare cu taskleții în Linux
  - Rulează la nivelul IRQL_DISPATCH_LEVEL
- Asynchrounous Procedure Call (APC)
  - nu este o acțiune amânabilă (în sensul că nu face parte din ciclul de tratare al unei întreruperi)
  - rulează la nivelul IRQL_APC_LEVEL
  - oferă posibilitatea rulării de operații în contextul unui anumit proces
  - sunt folosite pentru a termina operațiile IO asincrone (în kernel toate operațiile IO sunt asincrone)

Cum folosim DPC-urile?

- Inițializare: KeInitializeDpc()
- Activare: KeQueueDpc()
- Mascarea: se poate face prin ridicarea nivelului de întrerupere la nivelul DIRQL

Cum sunt implementate DPC-urile?

- La activare DPC-urile se introduc în coada DPC
- Când rulează?
  - la fiecare trecere DIRQL -> DPC
- Câte DPC-uri din coadă sunt executate?
  - toate
- Când se execută DPC-urile reactivate?
  - la o nouă trecere DIRQL -> DPC
  - din idle kernel thread

Intrebări

?