# 23. Tétel

Ismertesse a megszakítás kezelés implementációját a Linux kernelben! Térjen ki a Bottom Half mechanizmus bemutatására is!

## Megszakítások

Eszközvezérlő megszakításának lekezeléséhez kell:

* **megszakítás kezelő fv**
  + általános alakja: typedef irqreturn\_t (\*irq\_handler\_t)(int irq, void \*devid);
    - irq – megszakítás száma
* **regisztrálni kell a kezelő fv-t az adott megszakításhoz a driverben**
  + regisztráció: int rquest\_irq(unsigned int irq, irq\_handler\_t handler, unsigned long flags, const char \*devname, void \*devid),
    - handler – megszakítás kezelő fv
    - flags – opciók
      * IRQF\_DISABLED: gyors interrupt jelzése. Az implementációnak gyorsnak kell lennie, mert a megszakítás kezelése alatt a megszakítás le van tiltva
      * IRQF\_SHARED: az interruptot megosztjuk más kezelőkkel (pl. több HW használja ugyanazt a megszakítást)
      * IRQF\_SAMPLE\_RANDOM: a megszakítás felhasználható véletlen szám generálásáhz
      * IRQF\_TIMER: a megszakítás timer interrupt
    - devname – eszközvezérlő neve
    - devid – megosztott megszakítás esetén egyedi azonosító
    - visszatérési érték:
      * IRQ\_HANDLED: a megszakítást lekezelte a fv.
      * IRQ\_NONE: a megszakítást nem kezelte le a fv.
* **a driver eltávolításakor a regisztrációt is el kell távolítani**
  + törlés: void free\_irq(unsigned int irq, void \*devid);
  + a megszakítást több eszköz is használhatja!! 🡪 fontos a pontos devid

## Megszakítások megosztása

* **a megszakítás vonalak száma véges 🡪 lehetőség van egy vonalhoz több HW megszakításait is bedrótozni**
* a megszakítás kezelő fv. regisztrációjánál jelezni kell a megosztott használatot (IRQF\_SHARED)
* devid: egyedi érték, ami a regisztrációt azonosítja
* a megszakítás kezelő fv-nél lehet jelezni, hogy nem ennek a fv-nek szólt a megszakítás: IRQ\_NONE visszatérési értékkel
* megosztott megszakítások 🡪 egy adott vonal letiltása így nem csak egy eszközre van kihatással !!!

## Kezelő fv-ek megkötései

* nem használható sleep() 🡪 aktiválná a schedulert
* kmallock() csak GFP\_ATOMIC flaggel, mivel könnyen sleepbe mehet
* kernel és user space között nem mozgatunk adatokat
* gyorsaság
* a processz specifikus adatok nem elérhetőek a megszakítás kezelőből (mert nem processz hívta meg)

## Bottom Half mechanizmus

* ha komolyabb adatfeldolgozást kéne végrehajtani egy megszakítás következtében 🡪 nem lehet megcsinálni megszakításban, mert az fáj 🡪 solution: BH mechanizmus: felbontja a feladatot két részre
  + **Top half**
    - tényleges megszakítás kezelő rutin
    - gyors letárolás
    - kérvényezése a Bottom Half futásának minél gyorsabban
  + **Bottom half**
    - nem megszakításidőben futó rész
    - nem érvényesek a szigorú megkötések
* implementációjára megoldás: Kernelszálas megszakítás

## Kernelszálas megszakítás kezelés

### Működése

* regisztráció 🡪 a kernel létrehoz egy szálat az adott megszakítás lekezeléséhez
* megszakítás érkezik 🡪 handler elkapja, gyors feldolgozás 🡪 átadja a szálban levő függvénynek a paramétereket 🡪 a számolás már nem interruptban fog történni
* ha a szálban történő végrehajtás túl lassú 🡪 legrosszabb esetben az adott megszakítás kezelését akadályozza

### Használata

* regisztrálása: int request\_threaded\_irq(unsigned int irq, irq\_handler\_t handler, irq\_handler\_t thread\_fn, unsigned long flags, const char \*name, void \*dev);
  + handler – hagyományos megszakítást kezelő fv (tipikusan NULL, és akkor csak átadja a szállnak a megszakítás paramétereit)
  + thread\_fn – szál-kontextusban meghívandó fv.
* regisztrálás eszközhöz kötéssel 🡪 a felhasználás is együtt történik az eszközzel
  + int devm\_request\_threaded\_irq(struct device \*dev, unsigned int irq, irq\_handler\_t handler, irq\_handler\_t thread\_fn, unsigned long irqflags, const char \*devname, void \*dev\_id);
    - \*dev – eszközt leíró struktúrára mutató

## Megszakítások tiltása, engedélyezése

* megszakítás tiltása:
  + void disable\_irq(unsigned int irq);
    - megvárja a függvény lefutását, ha megszakításban van
  + void disable\_irq\_nosync(unsigned int irq);
    - azonnali kill
* ismételt engedélyezés
  + void enable\_irq(unsigned int irq);
* összes megszakítás tiltása:
  + void local\_irq\_disable();
* összes IRQ engedélyezése
  + void local\_irq\_enable();
* érdemes a IRQ-kat menteni, és újra betölteni a disable/enable előtt
  + void local\_irq\_save(unsigned long flags);
  + void local\_irq\_restore(unsigned long flags);