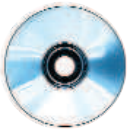
© Kiskapu Kft. Minden jog fenntartva

# Eszközosztályok



Árnyaljuk egy kicsit a képet, avagy hogyan írhatunk olyan eszközillesztőt, amelyik hibátlanul működik a 2.6-os rendszermag alatt.

A legutóbbi, „A rendszermag beállításai” című írásomban (Linuxvilág, 2003 júniusi szám) megismerkedtünk a rendszermag illesztőprogram-keretrendszerével, illetve az általános busz és az illesztőprogramok és -eszközök kódjának a működésével. A különböző alrendszerek működésének leírásakor az i2c magot hoztam fel példaként. Ebben a hónapban az illesztőprogram-osztálykód működése kerül terítékre, és ismét az i2c kódja szolgál példaként.
Mint a múlt alkalommal már szó esett róla, az eszközosztályok nem az objektumközpontú világ értelmezése szerinti osztályokat jelentik, inkább adott típusú szolgáltatást nyújtanak a felhasználó számára. A rendszermag osztályai például a tty, a blokkos vagy a hálózati eszközök és – hamarosan – a fájlrendszerek.
A 2.5.69-es rendszermagban teljesen újraírt illesztőprogram-osztályrendszer mutatkozott be. A korábbi rendszermagváltozatokban az osztályok támogatása szoros kapcsolatban állt az illesztőprogramok és az eszközök támogatásával. Az adott osztály egyben az eszközhöz is kötődött, amikor egy illesztőprogramnál bejegyzésre került. Ez sok eszköznél és osztálynál jól működött, de a valós élet bőségesen szolgált olyan eszközökkel, amelyek nem nagyon illettek ebbe a modellbe. Most az osztályok támogatása csak laza kapcsolatban áll az illesztőprogramokkal és az eszközökkel, eszközre vagy illesztőprogramra valójában már nincs is szükség az osztálykód használatához, mint ezt a tty-osztálykód is mutatja. Az osztálykód három különböző típusú szerkezetre osztható: osztályokra, osztályeszközökre és osztályfelületekre.

## Osztályok

A rendszermagban az osztályok megadása egy `struct class` adatszerkezettel történik. Igen, C-ben a class nem fenntartott szó. (Mindenki, aki C++ fordítóval szeretne rendszermagot fordítani, az új osztálykód készítőjéhez címezze jókívánságait.) Ha létre akarunk hozni egy osztály-adatszerkezetet, mindössze a `struct class` adatszerkezet `name` változójának kell értéket adnunk, és máris érvényes osztályt kaptunk. Ezt az alábbi kóddal tehetjük meg:

```
static struct class i2c_adapter_class = {
       .name = "i2c_adapter"
};
```

Az osztály adatszerkezetet megadás után a `class_register` függvény meghívásával lehet bejegyezni az illesztőprogram-magnál:

```
if (class_register(&i2c_adapter_class) != 0)
    printk(KERN_ERR "i2c adapter class
           ➥failed "
           "to register properly\n");
```

Ha a `class_register` függvény hibajelzés nélkül tér vissza, akkor a */sys/class/i2c_adapter* könyvtár sikeresen létrejött.

Később, ha az osztályt el kell távolítani, a `class_unregister` függvényt kell meghívni:

```
class_unregister(&i2c_adapter_class);
```

## Osztályeszközök

Az osztályok a különféle osztályeszközök kezelésére szolgálnak. A rendszermagban az osztályeszközök megadása a `struct class_device` adatszerkezettel történik. Ez az adatszerkezet számos, az illesztőprogram-mag által használt változót tartalmaz, ezekkel az illesztőprogram írójának nem kell foglalkoznia. Az általa értékkel ellátandó változók a következők:

- `class`: arra a `struct class` adatszerkezetre kell mutatnia, amelynek a feladata lesz az osztályeszköz kezelése.
- `dev`: az osztályeszközhöz hozzárendelt `struct device` adatszerkezet címét kell értékéül adni, amennyiben van ilyen. Egyetlen `struct device` adatszerkezetre több osztályeszköz-adatszerkezet is hivatkozhat. Ez a legfőbb különbség az előző rendszermag osztálytámogatása és a jelenlegi megvalósítás között. Ennek a változónak nem muszáj értéket adni, a rendszermag nélküle is jól működik. Ha mégis kap értéket, egy a `struct device` adatszerkezetre mutató közvetett eszközhivatkozás jön létre az osztályeszköz `sysfs` bejegyzésében. A példát lásd lejjebb.
- `class_id`: karaktertömb, az osztályeszköz leírását tartalmazza. Az adott osztály-adatszerkezethez hozzárendelt osztályeszköz-adatszerkezetek között egyedinek kell lennie.
- `class_data`: egy mutatót tartalmaz, amely az osztály-illesztőprogram által az osztályeszközhöz hozzárendelendő saját adatokra mutat. Ezt a változót közvetlenül nem kell elérni, értékének beállítására és lekérdezésére a `class_set_devdata` és a `class_get_devdata` függvény használható.

Megfelelően előkészített struct `class_device` adatszerkezet bejegyzését a `class_device_register` függvénnyel lehet elvégezni. Egy `struct class_device` adatszerkezet kezdeti értékadására és az illesztőprogram-magnál való bejegyzésére az alábbi, a ***drivers/i2c/i2c-core.c*** fájlból származó kódrészlet ad példát:

```
/* A csatoló hozzáadása az i2c_adapter
osztályhoz */

memset(&adap->class_dev, 0x00,
       sizeof(struct class_device));
adap->class_dev.dev = &adap->dev;
adap->class_dev.class = &i2c_adapter_class;
strncpy(adap->class_dev.class_id,
        adap->dev.bus_id, BUS_ID_SIZE);
class_device_register(&adap->class_dev);
```

Először a `struct class_device` változó (a `struct`

© Kiskapu Kft. Minden jog fenntartva

i2c_adapter változóba van beágyazva) kap kezdeti értékként nullát. Bejegyzés előtt minden illesztőprogrammodell-adatszerkezet összes változójának nullát kell értékként adni, az illesztőprogram-mag csak így tudja helyesen kezelni őket. Ezután a dev változó olyan értéket kap, hogy az i2c_adapter struct device változójára mutasson. Ebben az esetben a struct i2c_adapter adatszerkezet egy struct device és egy struct class_device adatszerkezetet is tartalmaz. A class változó az i2c_adapter_class változó címére fog mutatni, majd a class_id változó az eszköz bus_id azonosítójával megegyező értéket kap. Mivel az i2c_adapter eszköz bus_id azonosítója egyedi értékkel bír, egyben az i2c_adapter class_device adatszerkezetének class_id összetevője is egyedi lesz. Végül a class_device_register függvény meghívásával megtörténik az osztályeszköz-adatszerkezet bejegyzése a rendszermag eszközillesztő magjánál.
A fenti kód és két i2c csatoló betöltése után a próbagépen az *1. litstán* láthatóak szerint alakult a ***/sys/class/i2c_adapter*** fa.

```
$ tree /sys/class/i2c-adapter/
/sys/class/i2c-adapter/
|-- i2c-0
|   |-- device
    ➥-> ../../../devices/pci0/00:07.3/i2c-0
|   `-- driver ->
../../../bus/i2c/drivers/i2c_adapter
|-- i2c-2
    |-- device
    ➥-> ../../../devices/legacy/i2c-2
    `-- driver
    ➥-> ../../../bus/i2c/drivers/i2c_adapter
```

A fa alapján is látható, hogy az illesztőprogram-mag önműködően hoz létre egy eszköz- és egy illesztőprogram közvetett hivatkozást, és ezek a sysfs fa megfelelő pontjaira mutatnak. Ha a dev mutató nem a struct device adatszerkezetre mutat, akkor a közvetett hivatkozások nem jönnek létre. Ha bepillantunk a ***/sys/class/tty*** könyvtárba, akkor az osztályeszköz-bejegyzések túlnyomó részéhez nem tartozik struct device adatszerkezet, vagyis ezek a szimbolikus hivatkozások hiányoznak.

## Osztályfelületek

Az osztályfelületek révén tudja a rendszer értesíteni a saját kódunkat, ha egy struct class_device adatszerkezet egy adott osztálynál bejegyzésre, illetve eltávolításra kerül. Az osztályfelület megadása a struct class_interface adatszerkezettel történik, ami egészen egyszerű:

```
struct class_interface {
    struct list_head  node;
    struct class      *class;
    int (*add)        (struct class_device *);
    void (*remove)    (struct class_device *);
};*
```

A class változó értékeként kell megadni azt az osztályt, amelynek az eseményeiről értesülni szeretnénk. Az add (hozzáadás) és a remove (eltávolítás) változó értéke egy függvény lesz, amelyet akkor kell meghívni, ha eszközt adunk hozzá az osztályhoz, illetve távolítunk el belőle. Ha valamelyik eseményről nem akarunk értesülni, akkor a neki megfelelő változónak nem kötelező értéket adnunk.
Ha osztályfelületet szeretnénk bejegyezni a rendszermagnál, a class_interface_register függvényt kell használnunk, míg az eltávolítást a class_interface_unregister függvénnyel hajthatjuk végre. Az osztályfelületek használatára szép példát találunk a processzor órajel-magkódjában, amely a rendszermag forrásának ***kernel/cpufreq.c*** fájljában található meg.

## Fájlok létrehozása

Mint már említettem, az i2c-adapter osztály segítségével könnyen felismerhetjük a rendszerben lévő különféle i2c-csatolókat, illetve meghatározhatjuk helyüket az illesztőprogram-fában. Csakhogy az i2c-csatolókat a felhasználó nem címezheti meg közvetlenül. Amennyiben párbeszédbe szeretnénk elegyedni egy i2c-csatolóval, akkor be kell töltenünk egy i2c-lapka illesztőprogramot, vagy az i2c-dev illesztőprogramot kell használnunk. Az i2c-dev illesztőprogram karakteres illesztőfelületet biztosít a rendszerben jelenlévő összes i2c-csatolóhoz. Mivel nem árt pontosan tudni, hogy mely i2c-dev eszközök mely i2c-csatolókhoz csatlakoznak, létrehoztuk az i2c-dev osztályt:

```
static struct class i2c_dev_class = {
     .name    = "i2c-dev"
};
```

Miután az i2c-dev illesztőprogram az összes i2c-csatolót megtalálta, az eszközillesztő mag egy új i2c osztályeszközzel bővül. A hozzáadás az i2c_add_class_device függvényen belül történik:

```
static void
i2c_add_class_device(char *name, int minor,
                     struct i2c_adapter *adap)
{
    struct i2c_dev *i2c_dev;
    int retval;

    i2c_dev = kmalloc(sizeof(*i2c_dev),
GFP_KERNEL);
    if (!i2c_dev)
        return;
    memset(i2c_dev, 0x00, sizeof(*i2c_dev));

    if (adap->dev.parent == &legacy_bus)
        i2c_dev->class_dev.dev = &adap->dev;
    else
        i2c_dev->class_dev.dev =
        ➥adap->dev.parent;
    i2c_dev->class_dev.class = &i2c_dev_class;
    snprintf(i2c_dev->class_dev.class_id,
             BUS_ID_SIZE, "%s", name);
    retval =
        class_device_register
        ➥(&i2c_dev->class_dev);
    if (retval)
        goto error;
    class_device_create_file
    ➥(&i2c_dev->class_dev,
    ➥&class_device_attr_dev);

    i2c_dev->minor = minor;
    spin_lock(&i2c_dev_list_lock);
    list_add(&i2c_dev->node, &i2c_dev_list);
```

© Kiskapu Kft. Minden jog fenntartva

```
$ tree /sys/class/i2c-dev/
/sys/class/i2c-dev/
|-- i2c-0
|    |-- dev
|    |-- device
     ➥-> ../../../devices/pci0/00:07.3
|    `-- driver ->
../../../bus/pci/drivers/piix4-smbus
`-- i2c-2
     |-- dev
     |-- device
     ➥-> ../../../devices/legacy/i2c-2
     `-- driver
     ➥-> ../../../bus/i2c/drivers/i2c_adapter
```

```
    spin_unlock(&i2c_dev_list_lock);
    return;
error:
    kfree(i2c_dev);
}
```

A függvény csak néhány részletben tér el az `i2c_adapter` osztály bejegyzését végző kódtól. Először is a ***class_dev.dev*** mező vagy a csatoló szülőeszközére vagy magára a csatolóra mutat. Ennek oka az, hogy az i2c-csatolóknak valójában nincs szülőjük az általános rendszermag-eszközfában, hiszen vagy a rendszermag illesztőprogram-modelljében (például az ISA busszal ellentétben) nem szereplő buszon találhatók, vagy lényegében nem is csatlakoznak semmilyen buszra (mint például bizonyos beágyazott i2c-vezérlők). Ha egy i2c-csatolónak nincs helye a rendszermag eszközfájában, a rendszer az örökölt buszhoz rendeli hozzá. Az örökölt busz – a */**sys**/**devices**/**legacy*** alatt található – hivatott az összes ilyen eszköz kezelésére.
A második eltérést ennél az osztályeszköznél az alábbi sorban fedezhetjük fel:

```
class_device_create_file (&i2c_dev->class_dev,
➥&class_device_attr_dev);
```

A `class_device_create_file` függvény egy fájlt hoz létre az osztályeszköz könyvtárában. A fájlnevet és a jellemzőket a `CLASS_DEVICE_ATTR` makró adja meg az alábbiak szerint:

```
static ssize_t
show_dev(struct class_device *class_dev,
         ➥char *buf)
{
    struct i2c_dev *i2c_dev =
to_i2c_dev(class_dev);
    return sprintf(buf, "%04x\n",
           ➥MKDEV(I2C_MAJOR, i2c_dev->minor));
}
static
CLASS_DEVICE_ATTR(dev, S_IRUGO, show_dev,
                  ➥NULL);
```

A `CLASS_DEVICE_ATTR*` makró megadása a következő:

```
#define
CLASS_DEVICE_ATTR(_name,_mode,_show,_store)
```

```
$ tree /sys/bus/i2c/
/sys/bus/i2c/
|-- devices
|   |-- 0-0050 ->
../../../devices/pci0/00:07.3/i2c-0/0-0050
|   |-- 0-0051 ->
➥../../../devices/pci0/00:07.3/i2c-0/0-0051
|   |-- 0-0052 ->
➥../../../devices/pci0/00:07.3/i2c-0/0-0052
|   |-- 0-0053 ->
➥../../../devices/pci0/00:07.3/i2c-0/0-0053
|   `-- 2-0290 -> ../../../devices/legacy/
➥i2c-2/2-0290
`-- drivers
    |-- dev driver
    |-- eeprom
    |    |-- 0-0050 ->
➥../../../../devices/pci0/00:07.3/i2c-0/0-0050
    |    |-- 0-0051 ->
➥../../../../devices/pci0/00:07.3/i2c-0/0-0051
    |    |-- 0-0052 ->
➥../../../../devices/pci0/00:07.3/i2c-0/0-0052
    |    `-- 0-0053 ->
➥../../../../devices/pci0/00:07.3/i2c-0/0-0053
    |-- i2c_adapter
    `-- w83781d
         `-- 2-0290 ->
➥../../../../devices/legacy/i2c-2/2-0290
```

Az i2c-csatolókhoz tartozó */**sys**/**devices**/* könyvtárak tartalma:

```
$ tree /sys/devices/pci0/00:07.3
/sys/devices/pci0/00:07.3
|-- class
|-- device
|-- i2c-0
|    |-- 0-0050
|    |    |-- eeprom_00
|    |    |-- name
|    |    `-- power
|    |-- 0-0051
|    |    |-- eeprom_00
|    |    |-- name
|    |    `-- power
|    |-- 0-0052
|    |    |-- eeprom_00
|    |    |-- name
|    |    `-- power
|    |-- 0-0053
|    |    |-- eeprom_00
|    |    |-- name
|    |    `-- power
|    |-- name
|    `-- power
|-- irq
|-- name
|-- power
|-- resource
|-- subsystem_device
|-- subsystem_vendor
`-- vendor*
```

```
struct class_device_attribute
class_device_attr_##_name = {
     .attr   = {.name = __stringify(_name),
                .mode = _mode },
     .show   = _show,
     .store = _store,
};
```

A CLASS_DEVICE_ATTR* makrón belül az alábbi átadott értékeket találjuk:

- _name: a sysfs-en belül létrehozandó fájl neve, illetve része a name változónak, amely ezt a teljes jellemzőt írja le.
- _mode: a fájlhozzáférési mód, amellyel a fájl létrehozása történik. A megfelelő érték megadásához a szabványos hozzáférési makrók használhatók.
- _show: arra a függvényre mutat, amelynek meghívására a fájl tartalmának olvasásakor kerül sor. Ennek a függvénynek az alábbi visszatérési értékkel és átadott értékekkel kell rendelkeznie. A változónak nem kell értéket adni, ha a fájlból nem fogunk olvasni.

```
ssize_t
show (struct class_device *class_dev,
      ➥char *buf);
```

- _store: arra a függvényre mutat, amelynek a meghívására a fájlba való íráskor kerül sor. Ennek a függvénynek az alábbi visszatérési értékkel és átadott értékekkel kell rendelkeznie. A változónak nem kell értéket adni, ha a fájlba nem fogunk írni.

```
ssize_t
store (struct device *dev,
       ➥const char *buf, size_t count);*
```

Szinte minden illesztőprogram-modell adatszerkezete rendelkezik egy ATTR() makróval, amely megad egy fájlt a sysfs fán belül.
Ebben a példában egy ***dev*** nevű fájl jön létre a class_device_create_file függvény meghívásakor. A fájlból minden felhasználó csak olvashat. A fájl tartalmának olvasásakor az illesztőprogrammag a show_dev függvényt hívja meg. A show_dev függvény feltölti a neki átadott átmeneti tárat a felhasználónak továbbítandó adatokkal. Ebben az esetben a felhasználó az adott eszköz fő- és alszámát kapja meg. Minden fő- és alszámot használó osztályeszköznek rendelkeznie kell egy ***dev*** fájllal a megfelelő ***sysfs*** osztályeszköz könyvtáron belül.
A class_device_create_file függvény által létrehozott fájlok eltávolítására a class_device_remove_file függvény használható. Az eszközök eltávolításakor semmilyen fájlt nem kell kézzel törölni. Amikor egy eszközt eltávolítunk a sysfs alól, akkor a neki megfelelő könyvtárból a sysfs-mag önműködően törli az összes fájlt. Ha tehát eltávolítjuk az i2c-dev osztályeszközt a rendszerből, mindössze az alábbi parancsokra van szükség:

```
static void
i2c_remove_class_device(int minor)
{
     struct i2c_dev *i2c_dev = NULL;
     struct list_head *tmp;
     int found = 0;

     spin_lock(&i2c_dev_list_lock);
     list_for_each (tmp, &i2c_dev_list) {
          i2c_dev = list_entry(tmp, struct
                    ➥i2c_dev, node);
          if (i2c_dev->minor == minor) {
               found = 1;
               break;
          }
     }
     if (found) {
          list_del(&i2c_dev->node);
          spin_unlock(&i2c_dev_list_lock);
          class_device_unregister
                    ➥(&i2c_dev->class_dev);
          kfree(i2c_dev);
     } else {
     spin_unlock(&i2c_dev_list_lock);
     }
}
```

## Áttekintés

Ha az i2c-dev illesztőprogram és két i2c-csatoló illesztőprogram (az i2c-piix4 és az i2c-isa) be van töltve, akkor a */**sys**/**class**/**i2c-dev*** könyvtár tartalma a *2. listában* láthatók szerint alakul.

A */**sys**/**class**/**i2c-dev**/**i2c-2**/* könyvtárban lévő ***dev*** fájl tartalma az alábbi karakterlánc lesz:

```
$ cat /sys/class/i2c-dev/i2c-2/dev 5902
```

Ez a 86-os fő- és a 2-es alszámhoz tartozik, ezek az adott eszköz karakteres fő- és alszámai.
A */**sys**/**bus**/**i2c**/* könyvtár tartalma az alábbiak szerint alakul, miután néhány i2c ügyfél-illesztőprogramot betöltöttünk (*3. lista*). A 4. listát lásd az 53. CD Magazin/Eszköz könyvtárában.
A rendszermag illesztőprogram-modellje által alkalmazott, egymáshoz kapcsolódó adatszerkezet-mutatók rendszeréről, illetve felhasználói alkalmazásukról ***Jonathan Corbet*** készítette a legjobb leírást: „Web woven by a spider on drugs" (➲ http://lwn.net/Articles/31185/). Bízom abban, hogy két írásom alapján kicsit áttekinthetőbbé vált ez a hatalmas kuszaság, és sikerült valamennyire megismerkedni az eszközök között a rendszermagon belül fennálló kapcsolatokkal.

### Köszönetnyilvánítás

Szeretnék köszönetet mondani ***Pat Mochel***-nek, amiért ilyen kiváló és átfogó, a felhasználók számára is szemléletes keretrendszert készített az összes rendszermag-illesztőprogramhoz és eszközhöz. Ugyancsak hálával tartozunk mindenkinek, aki valamelyik rendszermagillesztőprogram-alrendszert tartja karban, és örömmel alakította át alrendszerét az új modellnek megfelelően. Segítségük nélkül az eszközillesztő kódja nem lett volna több tetszetős egyetemi feladványnál.

*Linux Journal 2003. augusztus, 112. szám*

***Greg Kroah-Hartman*** (greg@kroah.com)
Jelenleg a Linux USB és a PCI Hot Plug rendszermag felelőse. Az IBM-nél dolgozik, ahol számos, a Linux rendszermagjával kapcsolatos kérdéssel foglalkozik.

© Kiskapu Kft. Minden jog fenntartva