Egyszerű USB illesztőprogram írása

Az egész szobát beragyogó, színes fényű linuxos gép, és egyszerű USB illesztőprogram írása a következő szerzeményed számára. Hogy mi köze a kettőnek egymáshoz?

Cikksorozatom eddigi részeiben áttekintettem, hogyan készíthetünk különféle típusú rendszermag-illesztőprogramokat, és ismertettem a rendszermag-illesztőprogram felületeit – TTY, soros és I2C – valamint az illesztőprogrammag működését. Itt az ideje, hogy továbblépjünk, és valódi eszközökhöz írjunk valódi illesztőprogramokat. Első célunk egy egyszerű, USB-s lámpa Linux alatti beüzemelése lesz.
Szerkesztőnk, ***Don Marti*** egy jópofa apróságot szemelt ki: ez a Delcom Engineering által készített USB Visual Signal Indicator. Don feladatomul azt tűzte ki, hogy bírjam működésre a készüléket Linux alatt.

Az eszközprotokoll

Ha egy eszközhöz illesztőprogramot szeretnénk írni, akkor először azt kell kitalálni, hogyan tudjuk vezérelni azt.
A Delcom Engineering egészen nagyvonalú, termékeihez teljes USB protokoll leírást mellékelnek, amely a hálózatról is ingyenesen letölthető. A leírásban megtaláljuk, hogy az USB vezérlőlapka milyen parancsokat fogad, és hogy hogyan kell azokat használni.
Egy Microsoft Windows DLL-t is adnak, amelynek alapján más operációs rendszerek alá is könnyebben elkészíthető a készüléket vezérlő program.
Az eszközhöz mellékelt leírás ugyanakkor csak a benne található USB-vezérlő működését tárgyalja. A különféle színűLED-ek kapcsolgatását nem ismerteti, ezt magunknak kell kitalálnunk. A készülék kinyitása után – vigyázzunk, nehogy a csavarok eltávolításakor elrepüljön a belül lévő kis rugó – vizsgáljuk meg a belsejében található nyomtatott áramkört.
Ellenállásmérő vagy bármilyen más, zárt áramkör felismerésére használható műszerrel kideríthetjük, hogy a három LED a fő vezérlőlapka 1-es kapujának első három lábára csatlakozik-e. Ha megnézzük a leírást, akkor megtudjuk, hogy az 1-es kapu lábain megjelenő szintek vezérlésére a `Major 10`, `Minor 2`, `Length 0` parancs szolgál. A parancs az USB parancscsomag legkisebb helyiértékű bájtját írja ki az 1-es kapura, amelynek alapállapotba hozatalkor az alapértelmezett szintje magas. Megvan tehát, hogy a különféle LED-ek beállításához milyen USB parancsot kell küldenünk a készüléknek.

Melyik LED melyik?

Már tudjuk, hogy a kapu lábait melyik paranccsal lehet engedélyezni, de azt még nem, hogy melyik színes LED melyik lábhoz csatlakozik. Egy egyszerű programmal ezt is kideríthetjük, amivel a kapu lábaihoz az összes lehetséges értékkombinációt előállítjuk, majd kiküldjük ezeket a készüléknek. Egy ilyen program segítségével állt elő az *1. táblázat*, amely az értékek és a LED-színek párosításait tartalmazza. Tehát, ha a kapu minden lába engedélyezve van (0x07 hexadecimális érték), akkor egyik LED sem világít. Ez összeillik a leírásban talált állítással, amely szerint alapállapotba hozatal után az 1-es kapu alapértéke magas. Valóban, nem sok értelme lenne annak, ha az első bekapcsoláskor bármelyik LED is világítana. Most már tudjuk, hogy adott LED bekapcsolásához a neki megfelelő lábat alacsony (kikapcsolt) szintre kell állítanunk. A táblázatból kiderül, hogy a kék LED-et a 2-es, a pirosat az 1-es, míg a zöldet a 0-s láb vezérli.

Rendszermag illesztőprogram

Újonnan szerzett információinkkal felvértezve gyorsan dobjunk is össze egy rendszermag illesztőprogramot!
Az már nyilvánvaló, hogy USB illesztőprogramot készítünk, de vajon milyen felületet vegyünk igénybe a felhasználói tér felé ? Blokk-eszközként elég furcsa volna egy ilyen lámpát kezelni – itt ugyanis nem adatok fájlrendszerbeli tárolásáról van szó –, ezért karakteres eszköz mellett döntünk.
Ha viszont karakteres illesztőprogramot készítünk, akkor egy fő- és egy mellékazonosítót kell fenntartanunk neki. Sőt, gondoljuk át azt is, vajon hány mellékazonosítót igényel az illesztőprogram? Mi történik, ha valaki 100 darab USB-s lámpát akar rákötni a gépére? Ha előrelátók akarunk lenni, akkor legalább 100 mellékazonosítót le kell foglalnunk, még ha ez súlyos pazarlásnak is tűnik mindazok szemében, akik egyszerre csak egy lámpát akarnak használni. Ha karakteres illesztőprogramot írunk, akkor arra is ki kell találnunk valamilyen módszert, hogy a különféle színeket külön-külön tudjuk kapcsolgatni. A karakteres illesztőprogramoknál ez hagyományosan különböző `ioctl` parancsokkal történik, de nekünk van egy jobb ötletünk is, minthogy új `ioctl` parancsokkal tűzdeljük meg a rendszermagot.

1. táblázat *A kapuértékek és a LED-színek párosításai*

Kapuérték (hexadecimális)	Kapuérték (bináris)	Bekapcsolt LED-ek
0x00	000	piros, zöld, kék
0x01	001	piros, kék
0x02	010	zöld, kék
0x03	011	kék
0x04	100	piros, zöld
0x05	101	piros
0x06	110	zöld
0x07	111	Egyik LED sem világít

Minden USB-s eszköz saját könyvtárral rendelkezik a *sysfs* fában, miért ne használjuk tehát a *sysfs*-t, és miért ne hozzunk létre három fájlt az USB eszköz könyvtárában, például ***blue(kék)***, ***red(piros)*** és ***green(zöld)*** névvel? Így bármilyen felhasználói térben futó programmal át tudjuk kapcsolni LED-es eszközünk fényeit, legyen szó C programról vagy héjparancsfájlról. Egyúttal megszabadulunk a karakteres illesztőprogram megírásának nyűgétől is, és mellékazonosítókat sem kell szerezni az eszközhöz.
USB-s illesztőprogramunkkal szemben alapvető elvárás, hogy biztosítsa az USB alrendszernek a következő öt dolgot:

- Egy mutató az illesztőprogram tulajdonos moduljára. Így az USB mag helyesen kezelni tudja az illesztőprogram modulhivatkozási számlálóját.
- Az USB illesztőprogram neve.
- Az illesztőprogram által kezelt USB azonosítók listája. Ezt egyrészt az USB mag használja annak meghatározására, hogy az adott eszközhöz melyik illesztőprogram társul, másrészt a felhasználói térben futó parancsfájlok ennek alapján töltik be az illesztőprogramot, amikor a felhasználó csatlakoztatja az eszközt a géphez.
- Egy `probe()` függvény, amelyet az USB mag hív meg, ha az USB azonosítók táblázata alapján egyező eszközt talált.
- Egy `disconnect()` függvény, amelynek meghívására az eszköz rendszerből való eltávolításakor kerül sor.

Az illesztőprogram mindezeket az adatokat az alábbi kódrészlettel teszi hozzáférhetővé:

```
static struct usb_driver led_driver = {
      .owner =          THIS_MODULE,
      .name ="usbled",
      .probe =          led_probe,
      .disconnect =     led_disconnect,
      .id_table =       id_table,
};
```

Az id_table változó megadása a következő:

```
static struct usb_device_id id_table [] = {
       { USB_DEVICE(GYÁRTÓAZONOSÍTÓ,
       ➥TERMÉKAZONOSÍTÓ) },
       { },
};
MODULE_DEVICE_TABLE (usb, id_table);
```

A `led_probe()` és a `led_disconnect()` függvényekről később lesz szó.
Az illesztőprogram moduljának betöltésekor a fenti `led_driver` adatszerkezetet be kell jegyezni az USB magnál. Ehhez mindössze az `usb_register()` függvényt kell meghívni:

```
retval = usb_register(&led_driver);
if (retval)
       err("usb_register sikertelen. "
           "Hibakód: %d", retval);
```

Amikor pedig az illesztőprogramot eltávolítjuk a rendszerből, akkor törölni kell bejegyzését az USB magnál:

```
usb_deregister(&led_driver);
```

A `led_probe()` függvény meghívására akkor kerül sor, amikor az USB mag megtalálja a lámpát. Ilyenkor csak annyit kell tennie, hogy elvégzi a készülék kezdeti értékadását és a megfelelő helyen létrehozza a három *sysfs* fájlt.
Mindezt az alábbi kód végzi el:

```
/* A helyi eszköz adatszerkezet kezdeti értékadása */
dev = kmalloc(sizeof(struct usb_led), GFP_KERNEL);
memset (dev, 0x00, sizeof (*dev));

dev->udev = usb_get_dev(udev);
usb_set_intfdata (interface, dev);

/* A három sysfs fájl létrehozása az USB
eszköz könyvtárában */
device_create_file(&interface->dev,
                  ➥&dev_attr_blue);
device_create_file(&interface->dev,
                  ➥&dev_attr_red);
device_create_file(&interface->dev,
                  ➥&dev_attr_green);

dev_info(&interface->dev,
    "Az USB LED eszköz csatlakoztatása
    ➥megtörtént.\n");
return 0;
```

A `led_disconnect()` függvény ugyanilyen egyszerű, hiszen csak fel kell szabadítanunk a lefoglalt memóriát és el kell távolítanunk a *sysfs* fájlokat:

```
dev = usb_get_intfdata (interface);
usb_set_intfdata (interface, NULL);

device_remove_file(&interface->dev,
                  ➥&dev_attr_blue);
device_remove_file(&interface->dev, &dev_attr_red);
device_remove_file(&interface->dev,
                  ➥&dev_attr_green);

usb_put_dev(dev->udev);
kfree(dev);

dev_info(&interface->dev,
```

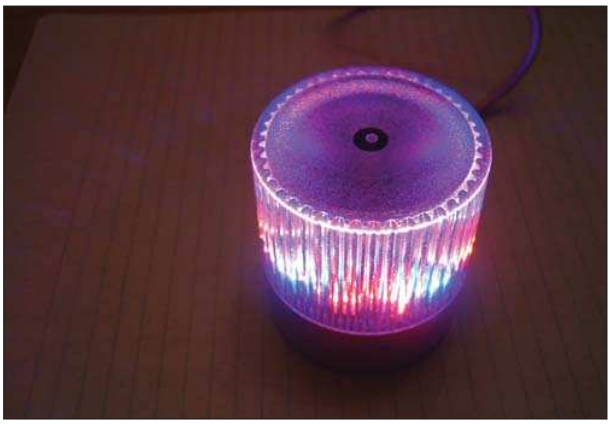
```
        "Az USB LED eszköz leválasztása
        ➥megtörtént.\n");
```

A *sysfs* fájlok olvasásával lekérdezhetjük a LED-ek aktuális állapotát, írásukkal pedig be- és kikapcsolhatjuk az egyes LED-eket. Ennek érdekében az alábbi makró mindegyik LED-hez két függvényt hoz létre, és megad továbbá egy *sysfs* eszközjellemzőfájlt:

```
#define show_set(value)
static ssize_t
show_##value(struct device *dev, char *buf)
{
   struct usb_interface *intf =
   ➥to_usb_interface(dev);
   struct usb_led *led = usb_get_intfdata(intf);

   return sprintf(buf, "%d\n", led->value);
}

static ssize_t
set_##value(struct device *dev, const char *buf,
          ➥size_t count)
{
   struct usb_interface *intf =
   ➥to_usb_interface(dev);
   struct usb_led *led = usb_get_intfdata(intf);
   int temp = simple_strtoul(buf, NULL, 10);

   led->value = temp;
   change_color(led);
   return count;
}

static DEVICE_ATTR(value, S_IWUGO | S_IRUGO,
                 ➥show_##value, set_##value);
show_set(blue);
show_set(red);
show_set(green);
```

Összesen hat függvényünk lesz, `show_blue()`, `set_blue()`, `show_red()`, `set_red()`, `show_green()` és `set_green()` névvel, továbbá három jellemző-adatszerkezet, `dev_attr_blue`, `dev_attr_red` és `dev_attr_green`. Mivel a *sysfs* fájlvisszahívók elég egyszerűek, illetve mindhárom érték esetében (blue-kék, red-piros és green-zöld) ugyanazt a műveletet kell elvégezni, makrót használtunk – így kevesebbet kell gépelni. Ez a megoldás nem szokatlan a *sysfs* fájlfüggvények körében, például a rendszermag forrásfájának I2C lapka illesztőprogramokat tartalmazó ágában, a *drivers/i2c/chips* könyvtárban is láthatunk ilyesmit. Nos, eljutottunk vére oda, hogy ha a felhasználó be szeretné kapcsolni a piros LED-et, akkor csupán egy 1-est kell írnia a *red* nevű *sysfs* fájlba. Ennek hatására meghívódik az illesztőprogram `set_red()` függvénye, amely viszont a `change_color()` függvényt hoza működésbe. A `change_color()` függvény így néz ki:

```
#define BLUE    0x04
#define RED     0x02
#define GREEN   0x01
   buffer = kmalloc(8, GFP_KERNEL);
```



A készülék a piros és a kék LED bekapcsolása után

```
color = 0x07;
if (led->blue)
   color &= ~(BLUE);
if (led->red)
   color &= ~(RED);
if (led->green)
   color &= ~(GREEN);
retval =
   usb_control_msg(led->udev,
               usb_sndctrlpipe(led->udev, 0),
               0x12,
               0xc8,
               (0x02 * 0x100) + 0x0a,
               (0x00 * 0x100) + color,
               buffer,
               8,
               2 * HZ);
kfree(buffer);
```

A függvény első lépésként a color (szín) változó összes bitjét 1-esre állítja. Ezután, ha valamelyik LED-et be kell kapcsolni, akkor átállítja a hozzá tartozó bitet. Következő lépésként USB vezérlőüzenetet küldünk a készüléknek, amiben kiírjuk rá a kívánt színbeállító értéket. Elsőre furcsának tűnhet, hogy kisméretű, mindössze nyolc bájtot tároló buffer változónkat `kmalloc` hívással hozzuk létre. Hogy miért nem adjuk meg egyszerűen a veremben, miért vesszük a nyakunkba a dinamikus helyfoglalás és felszabadítás nyűgét?
Nos, azért, mert a Linux futtatható néhány olyan géptípuson is, ahol a rendszermag-veremben létrehozott adatokat nem lehet USB eszköznek elküldeni, vagyis minden, USB-eszköznek szánt adatot dinamikusan kell létrehozni.

LED-ek működés közben

Megírtuk, lefordítjuk és betöltjük illesztőprogramunkat, és már gyönyörködhetünk is abban, ahogyan az kezelésbe veszi a gépünkhöz csatlakoztatott készüléket. Az illesztőprogramhoz kötődő USB eszközök mindegyike az illesztőprogram *sysfs* könyvtárában lelhető fel:

```
$ tree /sys/bus/usb/drivers/usbled/
/sys/bus/usb/drivers/usbled/
`-- 4-1.4:1.0 ->
```

```
../../../../devices/pci0000:00/0000:00:0d.0/usb4/
➥4-1/4-1.4/4-1.4:1.0
```

Az ebben a könyvtárban található fájl egy közvetett hivatkozás az USB eszköz valódi helyére a ***sysfs*** fában. Ha benézünk ebbe a könyvtárba, láthatjuk az illesztőprogram által a LED-ekhez létrehozott fájlokat:

```
$ tree /sys/bus/usb/drivers/usbled/4-1.4:1.0/
/sys/bus/usb/drivers/usbled/4-1.4:1.0/
|-- bAlternateSetting
|-- bInterfaceClass
|-- bInterfaceNumber
|-- bInterfaceProtocol
|-- bInterfaceSubClass
|-- bNumEndpoints
|-- blue
|-- detach_state
|-- green
|-- iInterface
|-- power
|   `-- state
`-- red
```

Ha most nullát vagy egyet írunk a blue, a green vagy a red fájlba, akkor megváltoznak a készülék fényei:

```
$ cd /sys/bus/usb/drivers/usbled/4-1.4:1.0/
$ cat green red blue
0
0
0
$ echo 1 > red
[greg@duel 4-1.4:1.0]$ echo 1 > blue
[greg@duel 4-1.4:1.0]$ cat green red blue
0
1
1
```

Ezzel a képen látható állapot áll elő.

Van másik út is?

Rendben, írtunk egy egyszerű rendszermag-illesztőprogramot (amely egyébiránt megtalálható a 2.6-os rendszermagfában, a ***drivers/usb/misc/usbled.c*** fájlban.) ehhez az aprósághoz, de vajon ez a legjobb módja a készülékkel való kapcsolattartásnak? Mi van az ***usbfs*** vagy a ***libusb*** használatával? Hiszen ezekkel külön illesztőprogram nélkül is vezérelhetjük készülékünket a felhasználói térből! Cikksorozatom következő részeiben erről lesz szó. Megmutatom, hogyan működik mindez, és néhány egyszerű héjparancsfájllal hogyan lehet vezérelni az USB-s lámpát.

Ha valaki látni szeretné, hogyan kell rendszermag illesztőprogramot írni valamilyen készülékhez, akkor ne habozzon, bátran tudassa velem.

Linux Journal 2004. április, 120. szám

Greg Kroah-Hartman