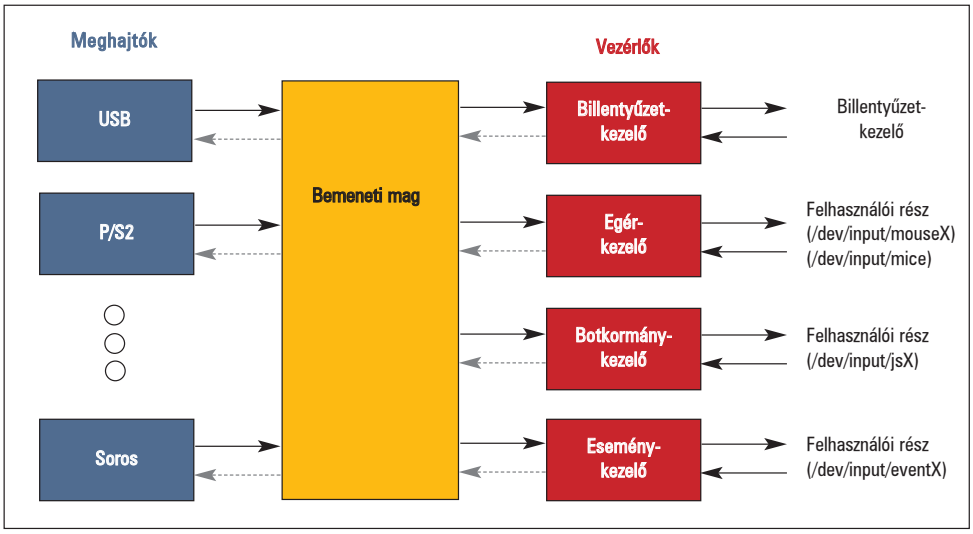
A linuxos USB alrendszer (1. rész)

Az USB bemeneti alrendszer egyre inkább terjed a magváltozatokkal. Épp itt az ideje megérteni, hogy pontosan mi is a dolga eszközeinkkel.

A linuxos USB alrendszer egyszerű, jól összehangolt módszer minden bemeneti eszköz kezelésére. A Linuxban ez egy viszonylag új megközelítés, ami részben megtalálható a rendszermag 2.4-es változataiban, és teljes egészében beépítették a rendszermag 2.5-ös fejlesztői változataiba. Ez a cikk négy alapterületet hivatott megismertetni: leírja, mit tesz pontosan a bemeneti alrendszer, bemutatja az alrendszer rövid történeti áttekintését, elmondja, hogyan épül fel a bemeneti alrendszer a rendszermagban, valamint bemutatja a felhasználói szintű API-t, vagyis az alkalmazásprogramozói felületet, és azt, hogy miként tudod ezt felhasználni programjaidban. Az első három témakörre még ebben a cikkben kitérünk, a felhasználói szintű API-t és a befejező témakört a következő rész tárgyalja.

Mi az a bemeneti alrendszer?

A bemeneti alrendszer a Linux-rendszermagnak az a része, ami a különböző bemeneti eszközöket kezeli (például a billentyűzetet, az egeret, a botkormányt, a digitáblát és sok más egyéb eszközt), ezek segítségével lép kapcsolatba a felhasználó a rendszermaggal, a parancssorral és a grafikus felülettel. Ezek az alrendszerek a rendszermagban találhatók, mivel elérésükhöz valamilyen különleges alkatrészcsatolót kell alkalmazni (mint például a soros kaput, a PS/2-es kaput, az Apple Desktop Bust és az Universal Serial Bust, vagyis az USB-t), amik nem érhetők el közvetlenül, és a rendszermag kezeli őket. Ezt követően a rendszermag egy általánosan összefüggő, az adott eszköznek megfelelő felületen keresztül teszi elérhetővé a kezelést a felhasználói programok számára, különböző programozói felületeken keresztül.



Kapcsolat a beviteli alrendszer részei között

Hogyan jutottunk el eddig?

A Linux bemeneti alrendszer elsősorban *Vojtech Pavlik* munkája, aki először a botkormányok támogatásának előkészítésekor érezte egy rugalmas bemeneti rendszer szükségességét, majd később is, amikor az USB-n kezdett el dolgozni. Az első sokrétű bemeneti alrendszer a már meglévő botkormány- és USB-meghajtókat támogatta a 2.3-as rendszermagok idején, ezt továbbvitték a 2.4-be, ahol ez a támogatás továbbra is a botkormányokra és az USB-re korlátozódik.

A 2.5-ös rendszermagban található bemeneti alrendszert már az összes bemeneti eszközre kiterjesztették. Cikkünk ezen az összetett rendszeren alapul, ami a 2.6-os rendszermagokban a bemeneti alrendszer programozói felülete lesz. Annak ellenére, hogy cikkünk írásakor a jelenlegi 2.4-es és 2.5-ös rendszermagok programozói felületei valamelyest eltérnek, folyamatos munka folyik, hogy ezek a különbségek megszűnjenek, amit többnyire a 2.4-es rendszermag frissítésével érnek el.

Pillantás a kulisszák mögé, avagy a rendszermag működésének vizsgálata

A bemeneti alrendszer három eleme: a bemeneti mag, az eszközmeghajtók és az eseménykezelők. A köztük lévő kapcsolat az *ábrán* látható. Amellett, hogy alapesetben az út az alacsony szintű eszközöktől az eszközmeghajtóig, az eszközmeghajtótól a bemeneti magig, a bemeneti magtól a kezelőig, a kezelőtől a felhasználói területre vezet, általában létezik valamilyen visszaút is. Ez a visszaút teszi lehetővé például a billentyűzet lámpáinak vezérlését és az erő-visszacsatolású (force-feedback) botkormányok mozgásának szabályozását. Mindkét irány ugyanazt az eseménymeghatározást alkalmazza, csak különböző típusú azonosítókkal.

A különböző részek együttműködését események vezérlik, amik szerkezetekként (structure) vannak meghatározva (*1. lista*). Az első mező, a ***time*** egy egyszerű időbélyeg, az ezt követő mezők már valamivel érdekesebbek. A ***type***, vagyis típus mező az esemény általános típusát jelöli, például, hogy az egéren nyomtunk-e le egy gombot vagy a billentyűzeten; de vonatkozhat a mozgásra is, mondjuk a botkormány esetében. A ***code*** mező mondja meg, hogy melyik gombot nyomták le pontosan, vagy azt, hogy az elmozdulás melyik tengely mentén történik; míg a ***value***, vagyis érték mező azt jelzi, hogy milyen a lenyo-

1. lista Az event-dev-struct.txt

```
struct input_dev {

    void *private;

    char *name;
    char *phys;
    char *uniq;
    struct input_id id;

    unsigned long evbit[NBITS(EV_MAX)];
    unsigned long keybit[NBITS(KEY_MAX)];
    unsigned long relbit[NBITS(REL_MAX)];
    unsigned long absbit[NBITS(ABS_MAX)];
    unsigned long mscbit[NBITS(MSC_MAX)];
    unsigned long ledbit[NBITS(LED_MAX)];
    unsigned long sndbit[NBITS(SND_MAX)];
    unsigned long ffbit[NBITS(FF_MAX)];
    int ff_effects_max;

    unsigned int keycodemax;
    unsigned int keycodesize;
    void *keycode;

    unsigned int repeat_key;
    struct timer_list timer;

    struct pm_dev *pm_dev;
    int state;

    int sync;

    int abs[ABS_MAX + 1];
    int rep[REP_MAX + 1];

    unsigned long key[NBITS(KEY_MAX)];
    unsigned long led[NBITS(LED_MAX)];
    unsigned long snd[NBITS(SND_MAX)];

    int absmax[ABS_MAX + 1];
    int absmin[ABS_MAX + 1];
    int absfuzz[ABS_MAX + 1];
    int absflat[ABS_MAX + 1];

    int (*open)(struct input_dev *dev);
    void (*close)(struct input_dev *dev);
    int (*accept)(struct input_dev *dev,
                  struct file *file);
    int (*flush)(struct input_dev *dev,
                 struct file *file);
    int (*event)(struct input_dev *dev,
                 unsigned int type,
                 unsigned int code,
                 int value);
    int (*upload_effect)(struct input_dev
         *dev, struct ff_effect *effect);
    int (*erase_effect)(struct input_dev
         *dev, int effect_id);

    struct list_head        h_list;
    struct list_head        node;
};
```

mott egérgomb vagy billentyű állapota, vagy azt, hogy milyen elmozdulásról van szó. Ha például a típus egérgomb vagy billentyű, a kód megmondja, hogy melyik egérgomb vagy billentyű az pontosan; az érték mező pedig elárulja, hogy a gombot éppen felengedték-e vagy pont lenyomták. Ehhez hasonlóan, ha a típus valamilyen viszonylagos tengelyt jelöl, a kód elárulja, hogy pontosan melyik tengelyről van szó, az érték mező pedig az elmozdulás nagyságát és irányát tartalmazza. Ha az egeret átlósan mozgatod, miközben ezzel egyidejűleg az egér görgőjét (scrolling-wheel) is mozgatod valamelyik ujjaddal, akkor minden frissítéskor három viszonylagos eseményt kapsz vissza: egyet a függőleges mozgásból (y tengely), egyet a vízszintes mozgásból (x tengely) és egyet a görgő mozgásából adódóan.
Az eseménykezelők szolgáltatnak felületet a felhasználói réteghez, olyan módon, hogy a szabványos eseményformátumokat úgy alakítják át, hogy azt a különböző programozói felületek megértsék. Általában ezek a kezelők gondoskodnak a különböző eszközcsomópontok (a */dev* bejegyzések) kezeléséről is. A legáltalánosabb kezelő a billentyűzet kezelője, ez az a szabványos bemenet, amivel a programozók (különösen a C-programozók) a leginkább tisztában vannak.
Az eszközmeghajtók általában az alacsony szintű alkatrészekkel tartják a kapcsolatot, mint például az USB-vel, a PCI-jal, a memóriával, a különböző ki- és bemeneti (I/O) területekkel és a soros kapu ki- és bemeneti területeivel. Ezek alakítják át a részegység által küldött felhasználói bemenetet az általános üzenet formájába, még mielőtt elküldik azt a bemeneti magnak. A bemeneti mag a rendszermag szabványos csatlakozóbővítmény (plugin) szerkezetét használja, az `input_register_device()` függvénnyel vesz fel új eszközöket, míg az `input_unregister_device()` függvénnyel távolít el meglévő eszközöket. Ezeket a függvényeket az `input_dev` kapcsolóval kell meghívni, ezt az első listában láthatjuk. Annak ellenére, hogy ez a szerkezet meglehetősen hosszúnak tűnik, a legtöbb bejegyzés arra szolgál, hogy az eszközmeghajtó beállíthassa, hogy az adott eszköz milyen lehetőségekkel bír, vagyis azt, hogy milyen típusú események és kódok fogadására vagy küldésére képes.
Azon túl, hogy a bemeneti mag kezeli az eszközmeghajtókat és a kezelőket, egy */proc* fájlrendszer felületet is létrehoz, amin keresztül látható, hogy éppen milyen eszközmeghajtók és kezelők működnek. Alább egy jellemző példa látható a */proc/bus/input/devices* fájlból, ami egy USB-s egér működését mutatja:

```
I: Bus=0003 Vendor=046d Product=c002
Version=0120
N: Name="Logitech USB-PS/2 Mouse M-BA47"
P: Phys=usb-00:01.2-2.2/input0
H: Handlers=mouse0 event2
B: EV=7
B: KEY=f0000 0 0 0 0 0 0 0 0
B: REL=103
```

Az I: sor tartalmazza az azonosító adatokat – a 3-as busztípus (ami az USB) és a gyártó, a termék és a változat adatait az egér USB-leíróiból. Az N: sor a nevet mutatja, amit úgyszintén az

USB eszközleírói tartalmaznak. A P: sor a fizikai eszköz adatait tartalmazza, ez esetünkben az USB-vezérlő PCI címéből, az USB-fából és a bemeneti eszközfájlból áll. Az `input0` azt jelöli, hogy ez az első logikai bemeneti eszköz az adott fizikai bemeneti eszközhöz. Néhány eszköz, mint például a multimédiás billentyűzetek, a fizikai eszköz egyes részeit rendelhetik az egyik logikai bemeneti eszközhöz, míg a fizikai eszköz más részeit egy másik bemeneti eszközhöz rendelhetik. A H: sor az eszközhöz rendelt kezelőt tartalmazza, erre később még kitérünk. Az egyes B: sorok a különböző bitmezőket takarják, amelyek az eszköz képességeit írják le – ebben az esetben az egyes egérgombokhoz tartozó billentyűket, a viszonylagos tengelyeket a golyóhoz és a görgőhöz.

A */proc* felület nagyon jól használható egyszerű eszközmeghajtók kipróbálására. Vegyük például azt az esetet, amelyben az eszközmeghajtó befűzéskor bejegyzi magát, illetve amikor az eszközt eltávolítjuk, az eszközmeghajtó is eltávolítja a hozzátartozó bejegyzést (*2. lista*). Ehhez néhány bevezető művelet elvégzésére van szükség az `init_input_dev()` függvénnyel. A függvény beállítja az eszköz nevét, a fizikai és azonosító leíróit, és beállítja a bittömböket, hogy jelezze: az eszköz képes bizonyos típusú események fogadására (az EV_KEY jelzi az egérgombokat és a billentyűket, két különböző kód a KEY_A és KEY_B jelzi a gombok címkéit). Az előkészítő eljárás ezt követően bejegyzi az eszközt a bemeneti magba. Ha ezt a kódot hozzáadod a rendszermaghoz (a `modprobe` paranccsal), látni fogod, hogy egy új eszköz jelenik meg a */proc/bus/input/devices* fájlban, mint az az alábbiakban is látható:

```
I: Bus=0019 Vendor=0001 Product=0001
Version=0100
N: Name="Example 1 device"
P: Phys=A/Fake/Path
H: Handlers=kbd event3
B: EV=3
B: KEY=10000 40000000
```

Ha valóban üzenetet szeretnénk küldeni az eszközmeghajtónktól a bemeneti magnak, előtte meg kell hívnunk az `input_event`-et vagy valamelyik kényelmes közvetítőt, például az `input_report_key`-t vagy az `input_report_abs`-t, amiket a ***linux/input.h***-ban találunk meg. A *3. listában* egy olyan kódot láthatunk, ami erre mutat példát. Ez a példa nagyjából ugyanazt mutatja, mint az előző, azzal a különbséggel, hogy itt egy időzítőt adunk hozzá, ami az `ex2_timeout()` függvényt hívja meg. Ez az új programrész négy KEY_A és négy KEY_B lenyomását továbbítja. A művelet hatására összes 16 esemény jön létre, mivel külön események keletkeznek a billentyűk lenyomásakor és elengedésekor is. Ezek az események továbbítódnak a bemeneti maghoz, azt követően a billentyűzetkezelőhöz, ami az „aaaabbbb” vagy az „AAAABBBB” betűsorok megjelenését eredményezni a kezelőfelületen vagy a parancssorban, a SHIFT billentyű állapotától függően. Az időzítő úgy állítódik be, hogy négy másodpercenként újra meghívódjon, egészen a végtelenségig. A négy másodperces várakozási idő azért lett beiktatva, hogy – ha eleged van a megjelenő karakterekből – el tudd távolítani a bővítményt. Ha csökkented a várakozási időt, győződj meg róla, hogy más módon is hozzá tudsz-e férni a rendszerhez, esetleg egy másik SSH-kapcsolaton keresztül. Ne felejtsd el meghívni az `input_sync` függvényt sem! Ez a függvény értesíti az eseménykezelőt (esetünkben a billentyűzetkezelőt), hogy az eszköz valamilyen belsőleg összefüggő adathalmazt küldött. A kezelő úgy is dönthet, hogy az eseményeket az `input_sync` meghívásáig tárolja.

2. lista A register.c

```
#include <linux/input.h>
#include <linux/module.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");

struct input_dev ex1_dev;

static int __init ex1_init(void)
{
    /* k l n sen  vatos indulÆs */
    memset(&ex1_dev, 0, sizeof(struct
           input_dev));
    init_input_dev(&ex1_dev);

    /* le r c mkØk beÆll tÆsa */
    ex1_dev.name = "Example 1 device";
    /* fut  rendszeren a phys egyedi*/
    ex1_dev.phys = "A/Fake/Path";
    ex1_dev.id.bustype = BUS_HOST;
    ex1_dev.id.vendor = 0x0001;
    ex1_dev.id.product = 0x0001;
    ex1_dev.id.version = 0x0100;

    /* ez az eszk z kØt kulccsal
       rendelkezik (A Øs B) */
    set_bit(EV_KEY, ex1_dev.evbit);
    set_bit(KEY_B, ex1_dev.keybit);
    set_bit(KEY_A, ex1_dev.keybit);

/* vØg l pedig bejegyezz k a bemeneti magba */
    input_register_device(&ex1_dev);

    return 0;
}

static void __exit ex1_exit(void)
{
    input_unregister_device(&ex1_dev);
}

module_init(ex1_init);
module_exit(ex1_exit);
```

Vessünk egy pillantást az utolsó eszközmeghajtó példánkra, ezúttal az után vizsgálódva, hogyan jönnek létre a viszonylagos adatok (*4. lista*). A példában egy olyan eszközmeghajtó látható, ami az egeret utánozza. A kezdeti beállítás szerint az eszköznek két viszonylagos tengelye van (REL_X és REL_Y) és egyetlen gombja (BTN_LEFT). Csakúgy, mint az előző példánkban, itt is egy időzítőt használunk, hogy meghívja az `ex3_timeout()` függvényt. Ez az időzítő ezt követően az `input_report_rel`-t hívja meg, hogy biztosítsa a viszonylagos mozgást (ötegységnyi lépésekkel – a viszonylagos elmozdulás a függvény harmadik értéke), ami harminc jobbra lépésből és harminc lefelé lépésből, valamint harminc balra lépésből és harminc felfelé lépésből áll, előidézve az egérmutató négyzetalakban történő mozgását. Hogy a mozgás érzését keltsük, az időtúllépést 20 milliszekundumra állítjuk. Jegyezzük meg, hogy az `input_sync`-et

3. lista Az aaaabbbb.c

```
struct input_dev ex2_dev;

void ex2_timeout(unsigned long
                 ➥unused/*UNUSED*/)
{
    int x;

    for (x=0;x<4;x++) {
      /* a vagy A betß */
      input_report_key(&ex2_dev, KEY_A, 1);
      input_sync(&ex2_dev);
      input_report_key(&ex2_dev, KEY_A, 0);
      input_sync(&ex2_dev);
    }
    for (x=0;x<4;x++) {
      /* b vagy B betß */
      input_report_key(&ex2_dev, KEY_B, 1);
      input_sync(&ex2_dev);
      input_report_key(&ex2_dev, KEY_B, 0);
      input_sync(&ex2_dev);
    }

  /* idız tı beÆll tÆsa nØgy mÆsodpercre */
    mod_timer(&ex2_dev.timer,jiffies+4*HZ );
}

static int __init ex2_init(void)
{

    ... elıkØsz tØs ...

    /* ismØtlıdı idız tı beÆll tÆsa */
    init_timer(&ex2_dev.timer);
    ex2_dev.timer.function = ex2_timeout;
    ex2_dev.timer.expires = jiffies + HZ;
    add_timer(&ex2_dev.timer);

    return 0;
}

static void __exit ex2_exit(void)
{
    del_timer_sync(&ex2_dev.timer);
    input_unregister_device(&ex2_dev);
}
```

mindig meg kell hívni, így biztosítva, hogy a bemeneti kezelők folyton összefüggő eseményeket kapjanak. Ez a követelmény az előző példánkban nem volt feltétlenül szükséges, de különösen fontos, ha olyan adatokat akarunk átadni a bemeneti magnak, mint például a viszonylagos mozgás, mert esetenként több tengelyre is szükség lehet a mozgás meghatározásához. Ha átlósan mozgatnád az egeret, ehhez hasonlót kellene megadnod:

```
...
input_report_rel(..., REL_X, ...);
input_report_rel(..., REL_Y, ...);
input_sync(...);
...
```

Ez biztosítja, hogy az egér átlósan mozogjon, ne pedig oldalra, majd fel.

Kezelők: a felhasználói szint

Ez előző szakaszban láthattuk, hogy az eszközmeghajtók az alkatrészek és a bemeneti mag között helyezkednek el. A részegység felől érkező eseményeket – ez többnyire a megszakításokat jelenti – bemeneti eseményekké alakítja át. A bemeneti események felhasználásához kezelőket használunk, amelyek a felhasználói szintű kapcsolódó felületet alkotják.
A bemeneti alrendszer tartalmazza mindazokat a kezelőket, amikre szükséged lehet: billentyűzetkezelőt a parancssor kezeléséhez, egérkezelőt az X Window System alatt futó alkalmazásokhoz, botkormánykezelőt a játékokhoz, valamint egy érintőképernyő-kezelőt is. Létezik egy általános célú kezelő is, az eseménykezelő, ami alapvetően a felhasználói területen futó programoknak a bemeneti eseményeit biztosítja. Ez azt jelenti, hogy szinte soha nincsen szükség rendszermagi kezelő írására, mivel ugyanezt a célt szolgálja az eseménykezelő és a hozzá tartozó felhasználói szintű kódok. Ezt a programozói felületet cikkünk második részében tárgyaljuk.

Köszönetnyilvánítás

Szeretném megköszönni ***Greg Kroah-Hartman*** és ***Vojtech Pavlik*** segítségét, amit a cikk első részének a megírásához nyújtottak.

A listák a 48. CD Magazin/USB könyvtárában találhatóak.

Linux Journal 2003. február, 107. szám

Brad Hards (bradh@frogmouth.net)
A Sigma Bravo technikai igazgatója egy szakértői szolgáltatásokat nyújtó kis cégnél, Canberrában. A Linux mellett repülőgéprendszerek összeillesztésével és minősítésével is foglalkoznak, csakúgy, mint GPS-szel és elektronikai hadieszközökkel.

KAPCSOLÓDÓ CÍMEK

A Linux bemeneti alrendszer elsősorban a 2.5-ös BitKeeper rendszermagban található meg. A BitKeeper sokoldalú rendszer, de ha csak a rendszermagot szeretnéd böngészni, a ➲ http://linus.bkbits.net:8080/linux-2.5 cím hasznos kiindulópontul szolgálhat.

Létezik egy kísérleti fejlesztési fa kizárólag a bemeneti alrendszerhez, amit a ➲ http://linux-input.bkbits.net:8080/linux-input címen érhetsz el.

Korábban a bemeneti alrendszert a ➲ http://linuxconsole.sourceforge.net címen lehetett elérni, de a rendszermaggal együtt átköltözött a BitKeeper alá. Bár az oldal nem tartalmaz túl sok leírást, található ott néhány hasznos folt a felhasználói szintű alkalmazásokhoz a CVS-fában.

Az AAAABBBB bővítmény egyik eredeti változata a ➲ http://www.wired.com/wired/5.08/linux_pr.html címen található meg.