A linuxos USB alrendszer (2. rész)

A bemeneti alrendszer használata során nem számít, hogy egy bemeneti eszköznek hány gombja van, vagy hogy hányféle esemény létrehozására képes – most már felhasználói szintből kezelheted őket.

Sorozatunk előző részében láthattuk, hogyan működik a Linux bemeneti alrendszer a rendszermagban, végül még az eseménykezelőkre is kitértünk. Valójában minden kezelő másfajta felhasználói szintű programozói felületet nyújt. A különböző bemeneti eseményeket egy bizonyos formátumba alakítják át, ami az adott programozói felület használatával érhető el.

A bemeneti alrendszer rendszermagba építésének egyik kulcspontja magának az eseményrétegnek a megléte. Az eseménykezelő a feldolgozatlan eseményeket karakteres eszközcsomópontokon keresztül juttatja el a felhasználói szintre – minden egyes logikai eszközre egy saját karakteres eszközcsomópont jut. Az eseménykezelő felület nagyon sokoldalú módszer, mivel a segítségével a felhasználói szinten lehetővé válik az események kezelése, anélkül, hogy bármiféle információ elveszne. Például az ősi egértípusok csak két tengelyt támogattak, és legfeljebb öt gombjuk lehetett; ezek ténylegesen két tengelyre lettek leképezve, és csak három valódi gombra, mivel a 4. és 5. gombokat a görgetőhöz tartozó fel és le irányoknak feleltették meg.

Ez azonban gondot okoz akkor, ha egy olyan egeret próbálunk meg használni, aminek háromnál több gombja van, és emellett még egy görgetővel is bír, mivel a további gombok csak a már létező gomboknak feleltethetők meg. A korábbi programozói felület a fejlettebb bemeneti eszközök használatát sem tette lehetővé. Ilyen eszköz például az űrgolyó, vagy bármilyen hozzá hasonló több tengelyű eszköz is. Ezzel szemben az eseményalapú programozói felülettel tetszőleges eszköz összes lehetőségét ki lehet használni. Az eseményalapú programozói felület eszközönkénti listával rendelkezik az eszközök képességeiről és jellemzőiről.

Írásunk az eseményréteg különféle `ioctl` képességeit elemzi, ami az alapvető írási és olvasási hívásokat egészíti ki.

1. lista Példa egy EVIOCGVERSION függvényre

```
/* Az ioctl() meghívja a szükséges
   eszközmeghajtót */
if (ioctl(fd, EVIOCGVERSION, &version)) {
    perror("evdev ioctl");
}

/* az EVIOCGVERSION ioctl() egy egész számot
 * ad vissza amit kicsomagolunk és
   megjelenítünk */
printf("evdev driver version is %d.%d.%d\n",
       version >> 16, (version >> 8) & 0xff,
       version & 0xff);
```

2. lista Az input_id szerkezet meghatározása

```
struct input_id {
    __u16 bustype;
    __u16 vendor;
    __u16 product;
    __u16 version;
};
```

3. lista Egy EVIOCGID ioctl

```
/* néhány eszközadat lekérdezése */
if(ioctl(fd, EVIOCGID, &device_info)) {
    perror("evdev ioctl");
}

/* Az EVIOCGID ioctl() input_devinfo
 * szerkezettel tér vissza -
 * (a <linux/input.h> fájlnak megfelelően)
 * Így végigmegyünk az egyes elemeken,
 * és megjelenítjük mindegyiküket.
 */
printf("vendor %04hx product %04hx version
➥%04hx",
       device_info.vendor,
       ➥device_info.product,
       device_info.version);
switch ( device_info.bustype)
{
 case BUS_PCI :
     printf(" is on a PCI bus\n");
     break;
 case BUS_USB :
     printf(" is on a Universal Serial
            ➥Bus\n");
     break;
...
```

Az eseményréteg változatszámának kiderítése

Az eseményréteg támogatja az esemény eszközkód-változatszámának lekérdezését az `EVIOCGVERSION` `ioctl` függvény felhasználásával. A függvény értéke egy 32-bites egész szám (`int`) típus, aminek a felső két bájtja a változatszám első részét, a harmadik bájtja a változatszám második részét, az alsó bájtja pedig a kiegészítő változatszámot tartalmazza. Egy számítógépen minden eseményalapú eszköz ugyanazzal az értékkel tér vissza. Az `EVIOCGVERSION` alkalmazására az *1. listában* láthatunk

4. lista Csonkított karakterláncok

```
int fd = -1;
char name[256]= "Unknown";

if ((fd = open(argv[1], O_RDONLY)) < 0) {
    perror("evdev open");
    exit(1);
}

if(ioctl(fd, EVIOCGNAME(sizeof(name)), name)
➥< 0) {
    perror("evdev ioctl");
}

printf("The device on %s says its name is
➥%s\n",
       argv[1], name);

close(fd);
```

5. lista Az EVIOCGPHYS és a kialakítási adatok

```
if(ioctl(fd, EVIOCGPHYS(sizeof(phys)), phys)
➥< 0) {
    perror("event ioctl");
}
printf("The device on %s says its path is
➥%s\n",
       argv[1], phys);
```

6. lista Egyedi azonosító megállapítása

```
if(ioctl(fd, EVIOCGUNIQ(sizeof(uniq)), uniq)
➥< 0) {
    perror("event ioctl");
}

printf("The device on %s says its identity is
➥%s\n",argv[1], uniq);
```

példát. Az `ioctl` függvény első értéke egy megnyitott fájlleíró az eseményalapú eszközcsomópontra (például a */dev/input/event0*-ra). Az `ioctl` függvény harmadik értékeként nem magát az egészszámtípust kell átadni, hanem egy rá hivatkozó mutatót.

Az eszköz azonosítójának megállapítása

Az `EVIOCGID` `ioctl` segítségével az eseményréteg lehetővé teszi a kapcsolódó eszköztől adatok lekérdezését. A függvény értéke egy mutató az `input_id` szerkezetre, ennek felépítése a *2. listában* látható. Az `__u16` kizárólag a Linuxra jellemző, előjel nélküli, 16 bites adattípust jelöl. Saját programjaidban az `__u16`-ot `uint16_t`-re alakítva is nyugodtan használhatod. A busz típusa az egyetlen olyan mező, amelyik pontos adatot tartalmaz. Megfontolandó, hogy ez a típus egy kötött, sorszámozott típust használjon, amit a `<linux/input.h>`-ban

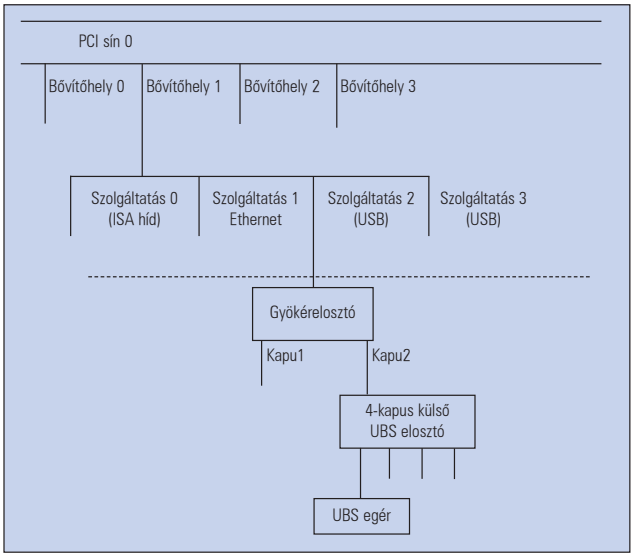
8. lista Események beolvasása

```
/* beolvasott bájtok száma */
size_t rb;
/* az események (egyszerre legfeljebb 64) */
struct input_event ev[64];

rb=read(fd,ev,sizeof(struct input_event)*64);

if (rb < (int) sizeof(struct input_event)) {
    perror("evtest: short read");
    exit (1);
}

for (yalv = 0;
     yalv < (int) (rb / sizeof(struct
     ➥input_event));
     yalv++)
{
    if (EV_KEY == ev[yalv].type)
        printf("%ld.%06ld ",
               ev[yalv].time.tv_sec,
               ev[yalv].time.tv_usec,
        printf("type %d code %d value %d\n",
               ev[yalv].type,
               ev[yalv].code,
               ev[yalv].value);
}
```



A billentyűzet topológiája

található `BUS_x` típus-meghatározókkal hasonlíthatunk össze. A ***gyártó***, az ***eszköz*** és a ***változatszám*** mezők az alkalmazott busz típusától függenek, az eszköz azonosítójának megfelelően. A jelenleg alkalmazott fejlett eszközök (főként az USB- és PCI-eszközök) ezeket az adatokat tartalmazzák, de az olyan örökölt eszközök, mint a soros egerek, a PS/2-es billentyűzetek vagy az ISA alapú hangkártyákon található játékkapuk (game ports) sajnos nem adnak erre vonatkozóan adatokat. Néhány busztípus esetében azonban ezek a számok értelmetlenek.

9. lista Egy írási függvény

```
struct input_event ev; /* the event */

/* első lépésben minden kijelzőt lekapcsolunk */
ev.type = EV_LED;
ev.code = LED_CAPSL;
ev.value = 0;
retval = write(fd, &ev, sizeof(struct input_event));
ev.code = LED_NUML;
retval = write(fd, &ev, sizeof(struct input_event));
ev.code = LED_SCROLLL;
retval = write(fd, &ev, sizeof(struct input_event));

while (1)
{
     ev.code = LED_CAPSL;
     ev.value = 1;
     write(fd, &ev, sizeof(struct input_event));
     usleep(200000);
     ev.value = 0;
     write(fd, &ev, sizeof(struct input_event));

     ev.code = LED_NUML;
     ev.value = 1;
     write(fd, &ev, sizeof(struct input_event));
     usleep(200000);
     ev.value = 0;
     write(fd, &ev, sizeof(struct input_event));
}
```

Az `EVIOCGID ioctl` használatára a *3. listában* láthatunk példát. Ez a függvény meghívja az `ioctl`-t, majd kiírja az eredményt. A `switch` szerkezet tartalmaz minden lehetséges busztípust; a program egy lehetséges kimenete: `vendor 045e product 001d version 0111 is on a Universal Serial Bus`. A busz típusán, a változatszámon és a gyártón (vendor), valamint az eszközt (product) jelölő számokon kívül néhány eszköz ezekhez tartozó karakterláncokat is ad, amelyek tartalmazhatják az eszköz és a gyártó pontos nevét. Ezeket a neveket az eseményrétegtől az `EVIOCGNAME ioctl` hívással lehet lekérdezni. Ez az `ioctl` egy karakterlánccal, és egy a karakterlánc hosszát jelölő egész számmal tér vissza (vagy egy negatív előjelű hibaazonosítóval). Ha a karakterlánc túl hosszú, a függvény levágja a karakterlánc végét. Az `ioctl` használatára a *4. listában* láthatunk példát. Talán feltűnt, hogy a függvénynek átadott kapcsoló nem `&név` formátumú. A magyarázat egyszerű: ha tömbre hivatkozunk, a tömb neve mindig egy mutató a tömb legelső elemére. Ha itt is `&név` alakban adnánk meg a tömböt, akkor valójában egy mutatót adnánk át, ami az első elemmutatóra mutat. Ha azonban mégis ragaszkodunk a `&név` formátumú megoldáshoz, akkor a `&(név[0])` alakot használhatjuk. Lássunk egy példát az eseménykód futtatására!
A */dev/input/event0*-ra befűzött eszköz elárulja a nevét: Logitech USB-PS/2 Optical Mouse.
Nem minden eszköz tartalmaz azonban használható neveket, ezért a rendszermagbeli bemeneti meghajtók valamilyen jelentéssel bírót próbálnak meg adni. Olyan USB-eszközök esetén, amelyek nem tartalmaznak a gyártóra vagy a készülék nevére utaló karakterláncokat, a bemeneti eszközmeghajtó összefűzi a gyártó és az eszköz azonosítóját, és azzal tér vissza.

Amellett, hogy az eszközazonosító és a névadat gyakran hasznos, olykor mégsem biztosítanak elegendő adatot arra nézvést, hogy milyen eszközzel is akadt dolgunk. Ha például két ugyanolyan botkormányod van, csak úgy tudod azonosítani őket, ha tudod, hogy melyik kapura csatlakoznak. Ezeket kialakítási adatoknak szokás nevezni, és az `EVIOCGPHYS ioctl` segítségével érhetők el.
Az `EVIOCGNAME`-hez hasonlóan ez is egy karakterlánccal tér vissza, valamint a hozzá tartozó hosszal (hiba esetén a negatív előjelű hibakóddal). Használatára az *5. listában* láthatunk példát.
A program lefuttatása a következőhöz hasonló kimenetet eredményez:

A /dev/input/event0-ra befűzött eszköz azt mondja, hogy az ő elérési útja usb-00:01.2-2.1/input0.

Hogy a fenti karakterláncot megérthessük, előtte darabokra kell szednünk. Az `usb` rész magától értetődően azt jelenti, hogy egy USB-eszközzel van dolgunk. A `00:01.2` érték az USB-vezérlő PCI-buszon lévő azonosítóját takarja, pontosabban a `00` számú a PCI-busz `01`-es rekeszének 2. feladata. A 2.1 a vezérlőtől az eszközhöz vezető utat mutatja, ahol a befűzött vezérlő az elsődleges vezérlő második rekeszébe kerül, az eszköz pedig a befűzött vezérlő második rekeszébe. Az `input0` azt jelenti, hogy az adott eszközön ez az első eseményeszköz. A legtöbb eszköz egyetlen ilyen eseményeszközzel bír, azonban például a multimédia-billentyűzetek a fő billentyűzethez tartozó eseményeket az első eseményeszközön továbbítják, a multimédia-művelethez tartozó eseményeket pedig egy második eseményeszközön adják tovább. Az ehhez tartozó példát az *ábrán* láthatjuk.
Ez az elrendezés nem nyújt megoldást, ha két azonos típusú eszközhöz tartozó kábelt cserélsz fel. Ebben az esetben csak akkor létezik megoldás, ha az eszközökhöz létezik valamilyen egyedi azonosítószám, például sorozatszám. Ezt az adatot az `EVIOCGUNIQ ioctl` hívással kérdezheted le, amire a *6. listában* láthatsz példát. A legtöbb eszköznek azonban nincs ilyen azonosítója, ilyenkor az `ioctl` üres karaktersorozattal tér vissza.

Az eszköz képességeinek és tulajdonságainak megállapítása

Néhány alkalmazás esetén elegendő az eszközazonosítót ismerni, mivel így – az eszköz fajtájától függően – minden eshetőség kezelésére képes lennél, a méretezhetőség szempontjából viszont ez nem megfelelő. Vegyük például, hogy engedélyezni szeretnéd a görgő használatát, de csakis abban az esetben, ha az egér rendelkezik görgővel. Nem lenne jó, ha a görgőt használó programunkban minden olyan gyártót és egértípust felsorolnánk, amelyik görgővel rendelkezik.
Az eseményréteg e nehézség elkerülését azzal teszi lehetővé, hogy megengedi annak megállapítását, hogy egy adott eszköz milyen képességekkel és jellemzőkkel bír. Az eseményréteg által támogatott tulajdonságok a következők:

- `EV_KEY` – meghatározott bináris értékek, például a billentyűk és az egérgombok.
- `EV_REL` – viszonylagos értékek, például az egér tengelyének elmozdulása.
- `EV_ABS` – meghatározott egészszámértékek, például a botkormány vagy a digitábla tengelyei.

10. lista Az eszköz gombjainak és billentyűinek mindenkori állapota

```
uint8_t key_b[KEY_MAX/8 + 1];

memset(key_b, 0, sizeof(key_b));
ioctl(fd, EVIOCGKEY(sizeof(key_b)), key_b);

for (yalv = 0; yalv < KEY_MAX; yalv++) {
   if (test_bit(yalv, key_b)) {
        /* a bit be van kapcsolva */
        printf("  Key 0x%02x ", yalv);
        switch ( yalv)
            {
            case KEY_RESERVED :
                printf(" (Reserved)\n");
                break;
            case KEY_ESC :
                printf(" (Escape)\n");
                break;
            /* other keys / buttons not
               shown */
            case BTN_STYLUS2 :
                printf(" (2nd Stylus
                ➥Button )\n");
                break;
            default:
                printf(" (Unknown key)\n");
            }
   }
}
```

- EV_MSC – egyéb dolgok, amelyek nem férnek bele egyetlen másik csoportba sem.
- EV_LED – a LED-ek és a hasonló kijelzők.
- EV_SND – hangkimenet, például a hangszórók.
- EV_REP – engedélyezi a billentyűk önműködő ismétlését a bemeneti magban.
- EV_FF – erő-visszacsatoló hatások küldése az eszköznek.
- EV_FF_STATUS – az eszköz által visszaadott jelentés az erő-visszacsatoló hatásokról.
- EV_PWR – energiagazdálkodási események.

Ezek a képességtípusok. Minden egyes típushoz további altípusok egész csoportja tartozik. Például az EV_REL különbséget tesz az X, az Y és a Z tengelyek, valamint a függőleges és vízszintes görgőtengelyek között. Hasonlóképpen az EV_KEY gombok és egérgombok százai között tesz különbséget.
Az eseményrétegen keresztül az EVIOCGBIT ioctl segítségével minden egyes eszköz tulajdonságai és képességei megadhatók. Ezzel a függvénnyel meghatározható, hogy egy eszköz milyen fajta tulajdonságokkal bír, például vannak-e billentyűi, gombjai, esetleg egyik sem. Ezen túlmenően az is meghatározható, hogy az eszköz pontosan milyen lehetőségeket támogat, például milyen billentyűk vagy gombok találhatók rajta.
Az EVIOCGBIT ioctl 4 kapcsolóval hívható meg, és a következőképpen néz ki:

```
ioctl(fd, EVIOCGBIT(ev_type, max_bytes), bitfield);
```

Az fd egy megnyitott fájlhoz tartozó fájlleírót jelöl, az

11. lista Az EVIOCGLED használata

```
memset(led_b, 0, sizeof(led_b));
ioctl(fd, EVIOCGLED(sizeof(led_b)), led_b);

for (yalv = 0; yalv < LED_MAX; yalv++) {
    if (test_bit(yalv, led_b)) {
        /* a bit be van kapcsolva */
        printf("  LED 0x%02x ", yalv);
        switch ( yalv)
            {
            case LED_NUML :
                printf(" (Num Lock)\n");
                break;
            case LED_CAPSL :
                printf(" (Caps Lock)\n");
                break;
            /* other LEDs not shown here*/
            default:
                printf(" (Unknown LED:
                ➥0x%04hx)\n",
                yalv);
            }
    }
}
```

ev_type a lekérdezendő tulajdonságok típusát adja meg (0-s típus esetén minden tulajdonságot lekérdezünk, nem csak az adott típushoz tartozókat). A max_bytes azt jelöli, hogy legfeljebb hány bájtnyi adattal térhet vissza a függvény, végül pedig a bitfield arra a területre mutat, ahova a lekérdezett adatokat a függvénnyel másoltatni szeretnénk. A függvény visszatérési értéke a bitfield területre ténylegesen átmásolt bájtok számát adja, vagy hiba esetén a hibakód negatív értékét.
Lássunk néhány EVIOCGBIT ioctl hívást tartalmazó példát. A 7. listában (49. CD Magazin/USB könyvtár) található kód például megmutatja, hogyan kérdezzük le az eszköz tulajdonságait. A kódban az evtype_bitmask méretét a *<linux/input.h>* fájlban található EV_MAX érték segítségével adjuk meg. Ezt követően kiadjuk az ioctl hívást, és az eseményréteg feltölti a bittömbünket. Ezután a tömb összes bitjét megvizsgáljuk, hogy lássuk, melyek vannak bekapcsolva, továbbá hogy az eszköz rendelkezik-e a tulajdonságok valamelyikével. A 2.5-ös rendszermagban minden eszköz támogatja az EV_SYN tulajdonságot, ezt a bitet a bemeneti mag kapcsolja be.
Ha a billentyűzet adatait kérdezzük le, akkor a következő eredményt kapjuk:

Támogatott eseménytípusok
- Eseménytípus 0x00 (összehangoló események)
- Eseménytípus 0x01 (billentyűk vagy gombok)
- Eseménytípus 0x11 (LED-ek)
- Eseménytípus 0x14 (ismétlés)

Az egér lekérdezése esetén a következőt láthatjuk:

Támogatott eseménytípusok
- Eseménytípus 0x00 (összehangoló események)
- Eseménytípus 0x01 (billentyűk vagy gombok)
- Eseménytípus 0x02 (viszonylagos tengelyek)

12. lista Az ismétlési beállítások lekérdezése: int rep[2];

```
if(ioctl(fd, EVIOCGREP, rep)) {
    perror("evdev ioctl");
}

printf("[0]= %d, [1] = %d\n", rep[0],
➥rep[1]);
```

13. lista Az ismétlés beállítása

```
int rep[2];

rep[0] = 2500;
rep[1] = 1000;

if(ioctl(fd, EVIOCSREP, rep)) {
    perror("evdev ioctl");
}
```

14. lista A letapogató kódok kiírása

```
int codes[2];

for (i=0; i<130; i++) {
    codes[0] = i;
    if(ioctl(fd, EVIOCGKEYCODE, codes)) {
        perror("evdev ioctl");
    }
    printf("[0]= %d, [1] = %d\n",
            codes[0], codes[1]);
}
```

15. lista A billentyűátrendezés

```
int codes[2];

codes[0] = 58; /* M keycap */
codes[1] = 49; /* assign to N */

if(ioctl(fd, EVIOCSKEYCODE, codes)) {
    perror("evdev ioctl");
}
```

Bemenet lekérdezése az eszköztől

Miután megállapítottuk, hogy egy eszköz milyen képességekkel rendelkezik, tudni fogjuk, hogy milyen eseményekre számíthatunk tőle és milyen eseményeket küldhetünk neki. Az események olvasásához csak egy egyszerű „olvasás” műveletet szükséges kezdeményezünk az eszközhöz tartozó karakteres eszközön. Minden alkalommal, ha az eseményeszközről olvasol (például */dev/input/event0*), egyszerre mindig események egész sorát fogod visszakapni, ahol minden esemény `input_event` szerkezetű.

A *8. listában* található kód egy ciklusban egy fájlleíróról olvas

17. lista A tengelyek mindenkori állapota

```
uint8_t abs_b[ABS_MAX/8 + 1];
struct input_absinfo abs_feat;

ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(abs_b)),
➥abs_b);

printf("Supported Absolute axes:\n");

for (yalv = 0; yalv < ABS_MAX; yalv++) {
    if (test_bit(yalv, abs_b)) {
        printf("  Absolute axis 0x%02x ",
                ➥yalv);
        switch ( yalv)
            {
            case ABS_X :
                printf("(X Axis) ");
                break;
            case ABS_Y :
                printf("(Y Axis) ");
                break;
            default:
                printf("(Unknown abs
                ➥feature)");
            }
        if(ioctl(fd, EVIOCGABS(yalv),
&abs_feat)) {
            perror("evdev EVIOCGABS ioctl");
        }
        printf("%d (min:%d max:%d flat:%d
        ➥fuzz:%d)",
                abs_feat.value,
                abs_feat.minimum,
                abs_feat.maximum,
                abs_feat.flat,
                abs_feat.fuzz);
      printf("\n");
    }
}
```

be eseményeket. Kiszűr minden olyan eseményt, ami nem billentyűkhöz tartozik, majd kiírja az `input_event` szerkezet egyes elemeit. A programot futtatása közben a billentyűzeten gépeltem, ami a következő kimenetet adta:

```
Esemény: időpont 1033621164.003838, típus 1, kód 37, érték 1
Esemény: időpont 1033621164.027829, típus 1, kód 38, érték 0
Esemény: időpont 1033621164.139813, típus 1, kód 38, érték 1
Esemény: időpont 1033621164.147807, típus 1, kód 37, érték 0
Esemény: időpont 1033621164.259790, típus 1, kód 38, érték 0
Esemény: időpont 1033621164.283772, típus 1, kód 36, érték 1
Esemény: időpont 1033621164.419761, típus 1, kód 36, érték 0
Esemény: időpont 1033621164.691710, típus 1, kód 14, érték 1
Esemény: időpont 1033621164.795691, típus 1, kód 14, érték 0
```

Minden egyes billentyűlenyomáshoz és felengedéshez külön esemény tartozik. Az eseményréteg olvasása a karakteres eszközök olvasásának jellemzőivel bír, vagyis egy ciklusban nem kell folyamatosan kiolvasnod az értékeket, kizárólag akkor, ha a programodnak az adott eszköztől valamilyen

bemenetre van szüksége. Ezen túlmenően, ha egyszerre több eszköz bemenetére is kíváncsi vagy, használhatod a `poll` és `select` függvényeket, hogy lásd, melyik eszközön van feldolgozható adat.
Eseményt ugyanolyan egyszerűen küldhetünk az eszköznek, mint ahogyan fogadunk tőle, azzal a különbséggel, hogy a `read` függvény helyett a `write` függvényt kell meghívnunk egy `input_event` szerkezetű eseménnyel. Erre a *9. listában* láthatunk példát. A példaprogram bekapcsolja a CAPS LOCK kijelzőjét, vár 200 milliszekundumot, majd lekapcsolja a kijelzőt. Majd elvégzi ugyanezt a NUM LOCK-kal, miután újból megismétli az első műveletet (egy végtelen ciklusban), így a billentyűzeten a két kijelző folyamatos villogását fogod tapasztalni.
Mostanra már világossá válhatott, hogy eseményeket csak akkor kapsz, ha valami történt – lenyomtak vagy felengedtek egy billentyűt, mozgatták az egeret stb. Néhány eszköz esetében ismerni kell az eszköz állapotát, például tudni szeretnéd, hogy egy billentyűzeten mely LED-ek vannak bekapcsolva, és melyek kikapcsolva, akkor is, ha a hozzájuk tartozó változást jelző esemény még a program indulása előtt következett be. Az `EVIOCGKEY ioctl` pontosan erre szolgál: lekérdezhetjük vele a billentyűk és gombok mindenkori állapotát. Használatára a *10. listában* láthatunk példát. Az `ioctl` nagyon hasonlatos az `EVIOCGBIT(...,EV_KEY,...)` függvényhez, de ahelyett, hogy a tömbben az eszköz gombjainak vagy billentyűinek a listáját küldené el, az `EVIOCGKEY` csak a lenyomott gombokhoz vagy billentyűkhöz tartozó biteket állítja be.
Az `EVIOCGLED` és az `EVIOCGSND` függvények megegyeznek a `EVIOCGKEY`-vel, azzal a kivétellel, hogy a bekapcsolt LED-ek, illetve a bekapcsolt hangok listájával térnek vissza. Az `EVIOCGLED` használatára a *11. listában* láthatunk példát. Tehát még egyszer: az `EVIOCGBIT` által kitöltött tömbben minden bitet hasonló módon kell értelmezni.
Az `EVIOCGREP ioctl` hívással a billentyűzet ismétlési beállításait kérdezheted le. Ennek működésére a *12. listában* láthatsz példát, ahol a tömbben két érték található. Az első érték a késleltetést határozza meg, még mielőtt a billentyűzet elkezdené az ismételt küldést, míg a második érték az ismétlések közötti várakozási időt adja meg. Ennek megfelelően, ha lenyomsz egy billentyűt, rögtön kapsz egy karaktert, a következő karaktert `rep[0]` milliszekundum múlva kapod, az azt követőt `rep[1]` milliszekundum múlva, míg az azt követő összes többit `rep[1]` milliszekundumonként, egészen addig, amíg fel nem engeded a billentyűt.
Ezeket a beállításokat az `EVIOCSREP ioctl` hívással változtathatod meg. A *13. listában* látható, hogy ez a hívás ugyanazt a kételemű tömböt használja, amit a lekérdezésnél is használtunk. A példaprogram a kezdeti várakozási időt 2,5 másodpercben határozza meg, míg az ismétlési időt 1 másodpercben.
Néhány bemeneti eszközmeghajtó támogatja a lenyomott billentyűk leképezett visszaadását (ahogyan azokat a billentyűzet érzékeli, és letapogató kódokat – scancode – ad vissza), és az eseményeket úgy küldi el a bemeneti rétegnek. Az `EVIOCGKEYCODE ioctl` hívással eldöntheted, hogy az egyes kódokhoz milyen billentyűk tartoznak. A *14. listában* lévő program az első száz letapogató kódon lépdel végig egy ciklusban. A letapogató kód értéke (a függvény bemenete) az egészszámtömb első mezője, és az ahhoz tartozó gombszám-esemény (a billentyűkód) a tömb második eleme.
Ezt a leképezést az `EVIOCSKEYCODE ioctl` hívással módosíthatod. A műveletet a *15. lista* szemlélteti, ahol az `ioctl` az M billentyűt az N-nek megfelelően képezi le, így az M minden egyes lenyomásakor egy N betűt kapsz vissza. Fontos, hogy a billentyűkódokat átállító `ioctl` függvények nem minden billentyűzeten működnek – például az USB-s billentyűzetek eszközmeghajtója nem támogatja az ilyen leképezések megadását.
Az `EVIOCGABS` ioctl is állapotadatokat ad vissza. De ahelyett, hogy egy bitmezőn jelölné be az egyes állapotokat, egy `input_absinfo` szerkezetet ad meg a rögzített tengelyre. Ha szükséged van az eszköz mindenre kiterjedő állapotára, akkor a függvényt meg kell hívnod minden létező tengelyre, mint az a *17. listában* látható. A tömbben lévő értékek előjeles, 32 bites mennyiségek, és nyugodtan kezelheted őket az `int32_t`-vel megegyező típusúként. Az első elem a tengely jelenlegi értékét adja meg, míg a második és a harmadik elemek a tengely jelenlegi határait jelöli ki, a 4. elem a válasz „flat" területének méretét szolgáltatja (ha van ilyen), míg az utolsó érték hiba esetén a hibához tartozó terület méretét adja meg.

Erő-visszacsatolás

Három további függvény létezik az erő-visszacsatolásos eszközök kezelésére: az `EVIOCSFF`, az `EVIOCRMFF` és az `EVIOCGEFFECT`. Ezek a függvények erővisszacsatolás-hatásokat küldenek, vonnak vissza, illetve megadják, hogy egyidejűleg hány hatás alkalmazható (ebben a sorrendben). Mivel az erő-visszacsatoláshoz tartozó programozói felület még fejlődik, illetve változásban van, egyelőre még korai lenne a teljes programozói felületet ismertetni. A *Kapcsolódó címek* rész tartalmaz olyan helyeket, amelyek talán már e cikk megjelenésekor is frissített adatokkal rendelkeznek a témakörben.

A cikkhez tartozó listák megtalálhatóak a 49. CD Magazin/USB könyvtárában.

Linux Journal 2003. március, 107. szám

Brad Hards
(bradh@frogmounth.net) A Sigma Bravo technikai igazgatója egy szakértői szolgáltatásokat nyújtó kis cégnél, Canberrában. A Linux mellett repülőgép-rendszerek összeillesztésével és minősítésével is foglalkoznak.

KAPCSOLÓDÓ CÍMEK

A Linux bemeneti alrendszer elsősorban a 2.5-ös BitKeeper rendszermagban található meg.
A BitKeeper nagyon sokoldalú rendszer, de ha pusztán a rendszermagot szeretnéd böngészni, a
➲ http://linus.bkbits.net:8080/linux-2.5 cím hasznos kiindulópontként szolgálhat.
Létezik egy kísérleti fejlesztési fa kizárólag a bemeneti alrendszerhez, amit elérhetsz a
➲ http://linux-input.bkbits.net:8080/linux-input címen.
Korábban a bemeneti alrendszer a
➲ http://linuxconsole.sourceforge.net címen volt megtalálható, de a rendszermaggal együtt átköltözött a BitKeeper alá. Bár az oldal nem tartalmaz túl sok dokumentációt, található rajta néhány hasznos folt felhasználói szintű alkalmazásokhoz.