Tanuljuk meg használni az ioctl rendszerhívásokat!

Eszközvezérlés ioctl hívásokkal.

Néhány évvel ezelőtt volt egy laptopom, amit a munkában és otthon egyaránt használtam. A hálózati csatlakozás megkönnyítése végett, illetve hogy ne kelljen helytől függően kézzel állítgatnom a dolgokat, úgy döntöttem, hogy mindkét helyen DHCP-t fogok használni. A munkahelyemen már megvolt, így hát otthon is felállítottam egy DHCP-kiszolgálót. A dolog jól működött egészen addig, amíg meg nem próbáltam hálózat nélkül elindítani a gépet. Ilyenkor ugyanis a gép rengeteg időt töltött el azzal, hogy – sikertelenül – megpróbálta megtalálni a DHCP-kiszolgálót, és csak azután folytatta az indítási folyamatot. Arra gondoltam, az lenne az eszményi megoldás, ha a rendszer ethernetcsatoló nélkül indulna, és csak akkor indítaná el a hálózatot, ha a kábel egy jelelosztóhoz (hub) csatlakozik, vagyis ha az ethernetkártyán világít a „link" (kapcsolat) lámpa. A legjobbnak az tűnt, ha írok egy olyan parancsfájlt, amely egy programocska visszatérési kódjából meg tudja állapítani, hogy az adott hálózati eszközön a „link" jelzés aktív-e. A feladat annak megállapítása volt, hogy miképpen lehet lekérdezni a 10/100Base-T kártyám kapcsolati állapotát.

Túl nagy gyakorlatom az alacsony szintű Linux-programozásban nem lévén eltartott egy ideig, amíg rájöttem, hogy az eszközvezérlőkkel folytatott ilyesfajta kapcsolatot általában az `ioctl` könyvtárhívásokon keresztül lehet megvalósítani (az `ioctl` az I/O control rövidítése), amelynek prototípusait a ***sys/ioctl.h*** állományban találjuk:

```
int ioctl(int, int, ...)
```

Az első érték a fájlleíró. Linux alatt minden eszközt fájlként érünk el, általában a célként kiválasztott eszközhöz tartozó fájlleírót használjuk fel. Az ethernetkártyák esetében azonban a fájlleíró egyszerűen egy nyitott foglalat (socket). Tulajdonképpen a foglalatot nem szükséges a kérdéses kártyához rendelnünk.

Az ***ioclt.h*** állományban található második érték a meghatározott `ioctl`-kérelemnek megfelelő egész szám. A kérelmek eszközről eszközre változnak: a soros eszközön például beállíthatjuk a sebességet, de a nyomtatón már nem. Természetesen a hálózati csatolókhoz is különleges parancskészlet tartozik.

A további értékek elhagyhatók és az egyik vagy másik eszköz `ioctl` megvalósításától függően változhatnak. Amennyire én tudom, a harmadik érték mindig jelen van, ugyanakkor háromnál többet sem láttam eddig.

A harmadik érték, úgy tűnik, általában egy szerkezetmutató. Ezáltal mindkét irányba tetszőleges mennyiségű adatot adhatunk át – a szerkezetben tárolt adatot egyszerűen a mutató átadásával továbbítjuk.

Az `ioctl` működésének egyszerű példáját az 1. listán bemutatott programban (lásd 59. CD-melléklet Magazin/ioctl könyvtárában) láthatjuk, amely egy adott jel állapotát ellenőrzi a soros kapun.

A program megnyitja a tty-eszközt (soros kapu), majd a soros kapu fájlleírójával, a `TIOCMGET` (azaz a modem helyzetbitjeinek lekérése) paranccsal és az eredmény tárolására szolgáló egész értékre mutató pointerrel meghívja az `ioctl` függvényt.

Ezután ellenőrizzük az `ioctl` eredményt, hogy megtudjuk, nem történt-e a kérés feldolgozása során valamilyen hiba. Amennyiben nem volt gond, ellenőrzésképpen a `TIOCM_DTR` tartalma és a visszaadott értékek közt logikai `ÉS` műveletet végzünk. Ez a lépés igaz vagy hamis, azaz nem nulla vagy nulla értéket adhat vissza.

Ethernetmeghajtókhoz ugyanígy használhatjuk az `ioctl` hívásokat. A foglalat alapú `ioctl` hívások (amelyeknél a fájlleíró valamely foglalat kezelője) harmadik értéke gyakran egy `ifreq` (interface request) szerkezetmutató. Az `ifreq` szerkezet típusmegadását (type deceleration) a ***net/if.h*** fájlban találjuk.

Számos `ioctl`-csatoló leírása sajnos csak igen nehezen lelhető fel, ráadásul a hálózati csatolók eléréséhez legalább három eltérő API létezik. A programot eredetileg a MII (Media Independent Interface) módszerrel írtam. A cikk írása közben – ahogy frissebb rendszermag került a gépemre – felfedeztem, hogy az `ETHTOOL` eljárást is fel kell vennem. Miután az `ETHTOOL`-t beépítettem, átalakítottam a programot és minden csatolófelület-eljárást külön alprogramba raktam át. A módosított program kipróbálja az egyik módszert, és ha nem megy, sorra veszi a következőt.

A harmadik módszer még a MII API-nál is régebbi, nem is találkoztam még olyan géppel, amelyiknek erre lett volna szüksége, így ezt a kódot nem építettem be.

Az MII csatolófelület használatával kapcsolatos adatokhoz elsősorban ***Donald Becker*** `mii-diag` nevű programjának (➲ ftp.scyld.com/pub/diag/mii-diag.c) kódelemzésével jutottam hozzá, amit a Scyld Computing Corporation honlapján találtam. Ezen a helyen egy kiváló oldalra bukkan (➲ http://www.scyld.com/diag/mii-status.html), ahol az `ioctl` által esetlegesen visszaadott MII állapot-

szavakkal kapcsolatos ismeretekhez férhetünk hozzá. Most azonban az ETHTOOL csatolófelületre fogok összpontosítani, hiszen ez a frissebb módszer.
Az ETHTOOL API használatával kapcsolatos ismereteket szintén a különféle forráskódok elemzésével gyűjtöttem – volt eset, hogy magukba a hálózati csatolók forráskódjába kellett belenéznem; különösképpen sokat segített az ***eepro100.c***. Ezenkívül igen hasznos volt ***Tim Hockin*** elektronikus levele, amire google-izás közben akadtam rá.
A programomat úgy készítettem el, hogy hacsak másként nem utasítjuk, alapértelmezettként az eth0 csatolófelületet használja. A csatolófelület ID az ifname változóban tárolódik. Minthogy az általam használt ioctl-parancsok csak erre a csatolóra érvényesek, más csatolót használva valószínűleg „cannot determine status” (az állapot nem meghatározható) jelzést kapunk.
Az ioctl meghívása előtt szükségünk lesz a fájlkezelőre, ezért először is meg kell nyitnunk egy foglalatot:

```
int skfd;
if (( skfd = socket( AF_INET, SOCK_DGRAM, 0 ) )
      ➥< 0 )
   {
   printf("socket error\n");
   exit(-1);
   }
```

A szokásos „próbáljunk meg minden hibát ellenőrizni” C-kódolási stílust híven követve az utasítást egy if feltételbe helyeztem, így ha a foglalat nem működik megfelelően, egyszerűen jelzi a hibát, majd -1 értéket visszaadva kilép a programból. Céljainknak talán jobban megfelel, ha az állapot meghatározhatatlanságát (hibát) a kapcsolat hiányának és nem kapcsolatnak értékeljük, ezért ha megtaláltuk a kapcsolatot 0-t, ha nem, 1-et adunk vissza.
A meghajtó eléréséhez biztosított új ETHTOOL API segítségével sokkal könnyebb volt megállapítani a kapcsolat állapotát, mint a korábbi módszerrel. Az ETHTOOL csatolófelületben úgy oldották meg az ioctl-kezelést, hogy csak egyetlen ioctl parancs létezik, a SIOCETHTOOL (ami azt mutatja meg, hogy az ETHTOOL parancs következik), majd az azt követő adatblokk tartalmazza az ETHTOOL-felület pontos alparancsát.
A szabványos ioctl adatszerkezethez (type ifreq) két további elemre is szükségünk lesz: a csatolófelület nevére, amelyre a parancsot alkalmazni szeretnénk, és a szerkezet címére (type ethtool_value), amelyben az adott parancsot és a visszaadott adatot tároljuk.
A szerkezeteket és az egyéb adatokat (ideértve a használható parancsokat is) az ***ethtool.h*** fejlécfájlban találjuk. Nekem az ETHTOOL_GLINK parancsra volt szükségem, amelyet „get link status” leírással láttak el – ezt helyeztem az ***edata.cmd*** mezőbe: edata.cmd = ETHTOOL_GLINK;
A csatolófelület nevét és az edata szerkezet címét el kell helyeznünk ifreq szerkezetben:

```
strncpy(ifr.ifr_name, ifname,
➥sizeof(ifr.ifr_name)-1);
ifr.ifr_data = (char *) &edata;
```

Innen már csak az maradt hátra, hogy meghívjuk a ioctl rendszerhívást, ellenőrizzük a visszaadott értéket (hogy lássuk, vajon a parancstípus engedélyezve volt-e), majd megvizsgáljuk a visszaadott mutató által meghatározott szerkezetet, ahonnan végül megtudjuk, hogy a kapcsolat élő-e vagy sem:

```
if (ioctl(skfd, SIOCETHTOOL, &ifr) == -1) {
    printf("ETHTOOL_GLINK failed: %s\n",
           ➥strerror(errno));
    return 2;
}
return (edata.data ? 0 : 1);
```

Jelen esetben a kód 0-t ad vissza, ha a kapcsolat él, 1-et pedig akkor, ha nem; illetve 2-t ad vissza, ha nem lehet eldönteni, esetleg hiba történt. A kód segítségével a függvényt az ***rc.local*** állományomból hívhatom meg, és ilyen módon a csatoló indítása után csak akkor indítom a dhcpcd-t vagy próbálkozom az IP-cím megszerzésével, ha a rendszer valamilyen működő jelelosztó vagy kapcsolóeszközhöz (hub, illetve switch) csatlakozik. Az ***rc.local*** idevágó részei a következőek:

```
/root/sense_link/sense_link | logger
if /root/sense_link/sense_link > /dev/null; then
  logger "No link sense -- downing eth0"
  /sbin/ifconfig eth0 down
else
  logger "Sensed link - dhcping eth0"
  /sbin/dhcpcd eth0
fi
```

Először is a sense_link kimenetét elküldjük a rendszernaplóba. Azután, ha az eth0 eszközön nem találtunk kapcsolatot, netán nem tudtuk megállapítani, mi a helyzet, üzenetet küldünk a naplóba, majd leállítjuk az eth0-t. Amennyiben kapcsolatot érzékeltünk, végrehajtjuk a dhcpcd-t az eth0 csatolón.
Amióta ezt a rendszert összeraktam, ***rc.local*** állományom igen gyorsan lefut, akár hálózati kábel nélkül, akár működő DHCP-kiszolgáló mellett használom. Az egyetlen eset, amikor lényeges várakozási idővel kell számolnom, akkor jelentkezik, amikor olyan hálózatra csatlakozom, ahol nincsen működő DHCP-kiszolgáló.
802.11b típusú kártyámmal még nem próbáltam ki a kódot, pedig kíváncsi vagyok, érzékeli-e a kapcsolatot a DHCP-kiszolgálóra fellépés előtt – én ugyanis általában csak a PCMCIA kártyát dugom be, ha olyan helyen járok, ahol tudom, hogy van kiszolgáló. Az érdekelt társaságnak azonban érdekes kísérlet és hasznos kiegészítés lenne.

Linux Journal 2004. január, 117. szám

Lisa Corsetti
Jelenleg programmérnök, egyben pedig az Anteil, Inc., egy különféle ipari és kormányzati területekre szánt, testreszabott, webalapú alkalmazásokra összpontosító cég elnöke. Lisa a Drexel Egyetemen szerezte elektromérnöki és informatikai diplomáját.