A telefonos rendszermag-API

Greg Herlein elmeséli, hogyan épül be a telefoneszközök meghajtóprogramja a Linux rendszermagjába.

Ritkaságszámba ment, még egy évvel ezelőtt is, az internetes telefonálás. Sokan gondolták, hogy nem is lesz ez jó soha igazi telefonhívások lebonyolítására. Ma, köszönhetően a Net2Phone, a Deltathree.com és a DialPad szolgáltatókhoz hasonlóaknak, amelyek Interneten keresztül bonyolított ingyenes vagy nagyon olcsó telefonhívásokat kínálnak, az IP-n keresztüli hangátvitel (VoIP) már majdnem húzóágazatnak számít. Bár ezekhez a szolgáltatásokhoz még nincsenek linuxos ügyfélprogramok, a Linux nincs lemaradva. A 2.2.14-es rendszermaggal a Linux nagy előnyt szerzett a számítógépes telefonálás területén: ez az első korszerű operációs rendszer, amely a rendszermag szintjén ad programozói felületet a telefonos alkalmazásokhoz. Ráadásul kiváló minőségű nyílt forráskódú programok már használják is ezt az API-t. Körbetelefonálhatjuk a világot a Linux és az Internet használatával – ingyen.

Ebben a cikkben bemutatjuk, hogyan építették be a telefoneszközök meghajtóprogramjait a rendszermagba oly módon, hogy egységes programozói felületet nyújtsanak a különféle típusokhoz. Ezután megtárgyaljuk az API tervezési és működési alapelveit. Mi módon választhatók szét az adatokra és az eseményekre vonatkozó adatok. Végül tárgyaljuk a telefonálás elemeinek (pl.: csengetés vagy a kézibeszélő felemelése) kezelését, ezt az úgynevezett „aszinkron eseményértesítő” folyamat végzi.

Új lehetőség a rendszermagban: a telefonálás támogatása

Sok ember kérdezte, miért kell egy új telefonálást támogató API-t a rendszermagba építeni. Még *Alan Coxot* is meg kellett győzni! Hiszen a legtöbb internetes telefonálásra alkalmas program képes kezelni a hangkártyát, és ha a hangkártya támogatja a teljes kétirányú (full-duplex) üzemmódot, és a felhasználónak van egy jó minőségű mikrofonos fejhallgatója, akkor a hívás minősége megfelelő lehet. Miért bonyolítsuk a dolgokat egy új API-val?

A válasz elég egyszerű: a hangkártyák nem telefonkészülékek! A hangkártyák nem tudnak vonalhangot adni, csörögni, foglaltat jelezni, vonalat bontani vagy hívó felet azonosítani, pedig ezek mindegyike szükséges a telefonkészülék rendes működéséhez. Egy hangkártya valóban használható némi korlátozással telefonálásra, ha egy mikrofonos fejhallgatót csatlakoztatunk hozzá. Az igazi telefonkártyákhoz azonban zsinór nélküli telefon is köthető, és máris megszabadulunk számítógéphez kötöttségünktől, amit a rövid zsinór okozott. Bonyolult üzleti telefonrendszerekhez is csatlakozhatunk. Ezenkívül a hangkártyákat nem lehet a helyi telefontársaság által felszerelt falicsatlakozóba bedugni, ezért nem tudjuk befolyásolni a kimenő és bejövő hívásokat, nem használhatjuk a díjszabáscsökkentő alkalmazásokat, vagy más, napjainkban íródó új nemzedékbeli telefonos alkalmazásokat.

Minden komoly Interneten keresztüli VoIP-megoldásnak tömöríteni kell a hangadatokat. Azért, hogy a megoldás megfeleltethető legyen a többi VoIP-alkalmazással és eszközzel, támogatnia kell a széles körben elfogadott tömörítőkodekeket, mint például a G.723.1 vagy a G.729a. Sajnos a programbeli megvalósítás esetén ki kell fizetni a kodekek felhasználási szerződésében kiszabott díjat, ez pedig akár hat számjegyű is lehet (dollárban). Ezek a könyvtárak nem lesznek nyílt forráskódúak, az biztos. A telefonkártyák (mint a lent említett Quicknet kártya) a gépünkbe beépítve tartalmazzák ezeket a kodekeket, ugyanis a felhasználási szerződésben kikötött díjat a kártya gyártója már rendezte. Ezzel elkerülhető a példányonkénti díj fizetése körüli hercehurca, hiszen a díj a kártya árába be van építve, és tetszés szerinti felhasználási szerződés alkalmazható a termékre, nem pedig csak az, mit a kodek szerzője előírt. Más szavakkal fogalmazva, írhatunk nyílt forráskódú VoIP-alkalmazást, és mégis használhatjuk a fejlett kodekeket. Nem túl gyakoriak azonban az olyan hangkártyák, melyek ismerik ezeket a tömörítőket.

1. lista A phone_device adatszerkezet

```
struct phone_device {
     struct phone_device *next;
     struct file_operations *f_op;
     int (*open) (struct phone_device *, struct
file *);
     struct file *file_p;
     int board;
     int minor;
};
```

Ráadásul a hangkártyák nem csatolhatók telefonokhoz vagy telefonrendszerekhez, és nem támogatják a kártyás hangtömörítőket. Az igazsághoz hozzátartozik, hogy a telefonkártyák viszont nem hangkártyák. A hangkártyák hangkeltő képességei magasan felülmúlják a telefonkártyákéit. Például a hangkártyák sztereók, a telefonkártyák monók. A hangkártyák képesek felvenni és lejátszani a zene tartományába eső frekvenciákat (20 Hz–20 kHz), a telefonkártyák azonban a beszédhangok frekvenciatartományára (általában 300 Hz–4 kHz) lettek korlátozva. A hangkártyák képesek bonyolult zenei hang keltésére, gyakran támogatják a MIDI-szabványt, és sokféle hanghatás létrehozható velük. A telefonkártyák mindezeket nem tudják. A felépítésük és a tulajdonságaik teljesen különbözők. Ebből következően az eszközmeghajtóiknak és az API-knak is különbözőnek kell lenniük. De várjunk csak! – mondhatnánk. – Mi a helyzet azokkal a nagyszerű programokkal, amelyek hangkártyával működnek, mint például a hangfelismerők vagy szövegfelolvasók? Nem lenne-e jó ezeket a telefoneszközön is használni csekély ráfordítással? Mindez lehetséges. A Linux telefonos API-ját úgy tervezték, hogy ne zárja ki eleve az olyan programok használatát a telefonkártyán, amelyeket eredetileg hangkártyához terveztek. A programkódot természetesen módosítani kell egy kicsit, de ez nem túl nehéz feladat. Erről több szó esik majd a cikk vége felé.

Az új API létrehozásának serkentésében a Quicknet Technologies Inc. járt az élen, mely többféle telefonkártyát is gyárt. Tavaly novemberben *Ed Okerson* és én (ekkor még a Quicknet alkalmazottja voltam) elküldtük a mai API ősét Alan Coxnak. Hetekig tartó megfeszített levelezés és számtalan programsor megírása után megszületett a Linux telefonos API-ja.

Vágjunk bele, lássuk, hogyan működik!

Az alapok: telefoneszközök

Az operációs rendszer szintjén minden eszköz egy szám. Ha belenézünk a /dev könyvtárba, sok fájlnevet láthatunk, de mélyen lent a Linux az eszközöket a fő- és alszámaik alapján azonosítja. Bizonyos típusú eszközök összessége egy közös főszámmal rendelkezik, az egyes eszközpéldányoknak ezen a típuson belül saját alszámmal kell bírniuk. Például ha kiadjuk az ls -al /dev/ttyS0 parancsot, a következőket láthatjuk:

```
gherlein@tux:~/lj > ls -al /dev/ttyS0
crwxrwxr-x   1 root      uucp        4,   64 Oct
       ➥27 06:23 /dev/ttyS0
```

Figyeljük meg, hogy a fájl jogosultságait leíró maszkban az első karakter egy "c", ez azt jelzi, hogy karakteres eszközről van szó. Az eszköz tulajdonosa a rendszergazda, és az uucp csoportban van. A következő két szám nem a fájl mérete, ahogy közönséges fájlok esetében lenne, hanem a fő- és alszám. A ttyS0 esetében a főszám 4 és az alszám 64.

A Linux telefonos API a telefonszerű eszközökhöz a 100-as főszámot rendeli hozzá. Elképzelhető, hogy az általunk használt Linux-változat nem hozza létre ezeket az eszközöket, ellentétben a /dev/ttyS*, a /dev/audio és más régebbi, széles körben elfogadott eszközleképezésekkel. H a /dev/phone* eszközök nincsenek meg a rendszerünkben, létre kell hoznunk őket a Linux telefonos API használata előtt. Könnyen megtehető ez saját kezűleg is a következő parancs kiadásával (rendszergazdaként):

```
mknod /dev/phone0 c 100 0
```

Ezzel egy új eszközfájlt hoztunk létre, amelynek a neve /dev/phone0. Ez egy karakteres eszköz 100-as főszámmal és 0-s alszámmal. Az mknod parancs részletes megismeréséhez olvassuk el a leírását. Gyakorlatilag csak annyi eszközfájlra van szükségünk, ahány fizikai eszköz csatlakozik a számítógéphez, de egyes emberek egyből létrehozzák az összes eszközfájlt 0-tól 15-ig.

Figyeljünk arra, hogy a /usr/src/linux/Documentation/devices.txt pillanatnyilag egy hibás kijelentést tartalmaz. Ebben a fájlban van felsorolva a főszámok hivatalos kiosztása, és jelenleg az szerepel itt, hogy a Linux a 159-et használja a telefoneszköz főszámaként, és a 100 már nem helyes. Ez egy elismert hiba, amit kijavítanak a közeljövőben. A helyes főszám a linuxos telefoneszközre (és a rendszermag által használt szám) a 100.

A phonedev modul – eszközközvetítés

Alan Cox fejlesztette ki a phonedev modult. A fejlesztés során hasonló megközelítéssel élt, mint amit a Video4Linux projekt során követett. Sok gyártó fog olyan terméket előállítani, ami telefoneszközként is használható. Ahelyett, hogy minden egyes telefoneszköz gyártója egy saját főszámot foglalna le, mindenki használhatja a 100 főszámot és a /dev/phoneN eszközfájlokat (ahol N az eszköz száma). Mindenkinek egy egységes, egyszerű programozói felületet kell nyújtania a felhasználói térben futó alkalmazások számára, azaz mindenkinek a közös API-t kell követnie, bár mindenki nyújthat kiegészítő szolgáltatásokat a termékéhez. A gyártók fogják megírni a saját eszközmeghajtó moduljaikat, amelyek megvalósítják a közös API-t és egy külső csatlakozási felületet az adott alkatrész belső részleteihez. A phonedev modul megoldotta azt a feladatot, hogy az aleszközszám futási időben képződjön le egy kártyagyártó által szállított modulra. A forráskód a /usr/src/linux/drivers/telephony/phonedev.c fájlban van, a phonedev.h és a telephony.h fejlécfájlokat pedig a /usr/include/linux könyvtárban találjuk. Lássuk, hogyan működik!

Minden telefoneszköznek egy phone_device adatszerkezetet kell használnia (lásd *1. lista*). Hasonlóképpen, minden telefoneszköznek meg kell hívnia két függvényt a phonedev modulban, hogy bejegyezze és törölje magát. Ezeket így adják meg:

```
extern int phone_register_device
     (struct phone_device *, int unit);
extern void phone_unregister_device
     (struct phone_device *);
```

Betöltéskor a phonedev modul beállítja magát és várja, hogy más telefoneszközöket kiszolgálhasson. Amikor egy telefoneszköz meghajtóprogramja betöltődik (a később tárgyalt modprobe-bal vagy insmoddal), meghívja a phone_register_device függvényt. Ennek leegyszerűsített működése: egy mutatót tart fenn a phonedev modulban, amely a phone_device adatszerkezetre mutat, megkeresi az első megnyitott alszámot és hozzárendeli azt a telefoneszközhöz, azután megnövel egy számlálót, amely azt követi nyomon, hogy valami használja a phonedev modult (nehogy használat közben törlődjön).

A gyakorlatban ez az alábbi következményeket vonja maga után: az először betöltődő telefonos modulhoz rendelődik hozzá az első elérhető (a legkisebb) alszám. Rendkívül fontos ezt megérteni abban az esetben, ha különböző gyártóktól származó moduloknak kell együtt élniük egy rendszerben, és kívánatos, hogy bizonyos eszköz egy meghatározott alszámon legyen elérhető (azaz egy adott /dev/phoneN eszközön keresztül). Más szavakkal, ha van egy ABC kártyánk és egy XYZ kártyánk, és azt szeretnénk, hogy az ABC legyen a /dev/phone0, akkor meg kell bizonyosodnunk róla, hogy az ABC meghajtóprogramja töltődik be először.

Minden eszköznek legalább az alapvető műveleteket értelmeznie kell, amelyekkel az eszközzel kapcsolatba léphetünk. Ilyen műveletek a megnyitás, az olvasás, az írás, a lezárás stb. Ezek mind a „fájlműveletek adatszerkezet” részei (lásd a linux/fs.h fejlécfájlt a részletekért). Minden eszköz úgy adja meg ezeket a függvényeket, ahogy számára megfelelő.

Ha egy program megnyitja a /dev/phoneN eszközt, akkor a phonedev modul fájlműveletek adatszerkezetében szereplő fopen függvényhívásra kerül a vezérlés. Ez a függvény a következő műveleteket hajtja végre:

- Megszerzi az aleszközszámot a /dev/phoneN fájlleíróból (inode).
- Összeállít egy karakterláncot a fő- és alszámok felhasználásával, ennek formátuma char-major-%d-%d. Például ha az alszám 0 (/dev/phone0 esetén), a karakterlánc char-major-100-0 lesz.
- Ezt a karakterláncot felhasználja a request_mmodule meghívásánál, mely egy modul betöltésére irányuló kérés. Ennek ugyanaz a hatása, mint a modprobe programnak (valójában egyszerűen elindít egy külön rendszermagszálat és ebben egy modprobe-ot). Ezzel lehetővé teszi, hogy – ha az eszköz egy modul és nem a rendszermag része – a kmod betölthesse a modult, mielőtt a phonedev használni próbálná.
- Ezután meghívja a telefoneszköz moduljának fopen függvényét, amely ténylegesen megnyitja a megfelelő eszközt.

Ahogy látható, a phonedev modulnak kettős szerepe van. Egyrészt az alszámokat betöltéskor dinamikusan hozzárendeli a telefoneszközökhöz, másrészt lehetővé teszi a telefoneszközök moduljainak betöltését futási időben. Ez világos, mégis jól kell ismerni a modprobe-ot és a modulok függőségi viszonyait ahhoz, hogy teljesen megértsük a telefonos modulokat és kölcsönhatásaikat.

A modprobe és a telefoneszközök

Ha a kmodot használjuk a szükséges rendszermagmodulok betöltéséhez, akkor létre kell hoznunk egy csomó másodnevet a /etc/modules.conf fájlban. Először is a rendszernek tudnia kell, hogy a char-major-100 a phonedev modul. Adjuk hozzá ezt a sort a fájlhoz:

```
alias char-major-100   phonedev
```

Ezután a rendszerrel közölni kell, hogy milyen telefoneszköz-modulok tartoznak az egyes alszámokhoz. Ahogy fent megtudhattunk, ez nem feltétlenül biztosítja azt, hogy a phonedev modul pont ezt az alszámot osztja ki a telefoneszköz-modulnak a betöltéskor. A következő példákban a Quicknet Technologies Inc. ➲ www.quicknet.net telefonkártyáit fogjuk használni. Ezekhez a kártyákhoz van meghajtóprogram a Linux rendszermagban, és viszonylag olcsóbbak, mint más telefonkártyák. A Quicknet eszközmeghajtója az ixj.o, és a modul neve ixj. Ez a meghajtóprogram az összes telefonkártyájukhoz használható (elég okos ahhoz, hogy kezelje az ISA, a PCI vagy a PCMCIA változatot és ismerje az egyes kártyatípusokon használt telefoncsatoló-áramköröket). Ha azt szeretnénk, hogy a Quicknet meghajtóprogramja a /dev/phone0 eszközhöz rendelődjön, akkor adjuk ezt a sort a /etc/modules.conf fájlhoz:

```
alias char-major-100-0  ixj
```

Ahogy emlékezhetünk még a phonedev fopen függvényének elemzéséből, a phonedev egy char-major-%d-%d formájú karakterláncot hoz létre, és a számok helyére a 100-at (főszám) és a kért alszámot helyettesíti be. Példánkban a /dev/phone0 megnyitási kísérlete azt eredményezi, hogy a phonedev megpróbálja betölteni a char-major-100-0 eszközt. Ez az eszköz ismeretlen a rendszermag modulbetöltője számára. A fenti alias parancs ezt a karakterláncot az ixj névre képezi le. Amikor megpróbáljuk megnyitni a /dev/phone0 eszközt, a phonedev modul automatikusan betölti az ixj modult, azután meghívja az ixj modul fopen függvényét, ez megnyitja az eszközt. (Természetesen feltettük, hogy a rendszermag támogatja a modulok betöltését, azaz `CONFIG_KMOD=y`).

Az egyszerű telefoneszköz-API

A Linux telefonos API-jának alapvető vívmánya, hogy telefoneszközre és a telefoneszközről csak hangadat írható, illetve olvasható az írás és az olvasás műveletek segítségével. Ez teljesen más – és másképpen is kezeli a meghajtóprogram –, mint az eseménytípusú adatok.

Hangadat

Mi is pontosan a hangadat, és miben különbözik az események adataitól? A hangadat a telefoneszköz mikrofonjáról származó hangjel analóg–digitális (A/D) átalakításának (és esetleg adattömörítésének) az eredménye. A telefon kézibeszélőjének felvételekor keletkező jel egy esemény. A sípszó, amelyet a készülék egy számbillentyűjének lenyomása idéz elő, szintén esemény, annak ellenére, hogy hangkibocsátással jár. Egy bejövő hívás által előidézett csengetés is esemény. Röviden minden, nem mikrofonon keresztüli bemenet esemény. Minden mikrofonon keresztül érkező bemenet (és a hangszórón megjelenő kimenet) hangadat. Ezt a hangadatot olvassuk és írjuk a telefoneszközről, illetve telefoneszközre a szabványos read és write rendszerhívásokkal.

A legtöbb telefoneszköz tömöríti a hangadatokat. Valójában az adattömörítés nélkül nem képzelhető el egy sikeres internetes telefonalkalmazás. Ezeket a tömörítési módszereket hívják „hangkodekeknek” vagy röviden „kodekeknek”. Vannak széles körben elterjedt és használt kodekek. A Linux telefonos API-ja tartalmaz állandókat ezekhez a kodekekhez, de egy adott telefoneszköz nem feltétlenül támogatja ezek mindegyikét. Egy ioctl rendszerhívással kérdezhető le, hogy milyen kodekek vannak használatban.

Az egyik nagy különbség a linuxos telefonos API és a hangkártyákhoz használatos API-k között az, hogy a linuxos telefonos API „adatkocka-központú”, míg a hangkártyák „bájtközpontúak”. Az adatkocka-központú eszköz időegység alatt egy meghatározott méretű adatkocka olvas be. Ez azért van így, mert minden hangkodek adott ideig dolgoz fel egy bizonyos ideig tartó hangadatot (általában 10, 20 vagy 30 ezredmásodpercnyi adatot). Mivel a tömörítés alkalmazása a bevett szokás a hálózati telefonos alkalmazásoknál, ezért ez a helyes és elvárt működése a telefoneszközöknek. A hangkártyáknál nincs ilyen megkötés, bármelyik megszólítás alkalmával tetszőleges számú bájtot írhatunk vagy olvashatunk. Az API meghatározza minden kodekhez az „adatkocka méretét”, és a nyers tömörítetlen hangadat is egy lehetséges kodekválasztásnak felel meg. Például a LINEAR16

2. lista A telefon csörgetése

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/telephony.h>
#include <linux/ixjuser.h>


int
main(int argc, char *argv[])
{
  int ixj=-1;
  char pname[80], maxrings;

  if(argc >= 2)
    sprintf(pname, "%s", argv[1]);
  else
    sprintf(pname, "/dev/phone0");

  printf("%s megnyitása\n",pname);
  ixj = open(pname, O_RDWR);

  if(ixj<1)
  {
    printf("Eszközmegnyitási hiba:
%s\n",pname);
    perror("open ");
    exit(-1);
  }

  if(argc >= 3)
    maxrings = atoi(argv[2]);
  else
    maxrings = 2;

  ioctl(ixj, PHONE_MAXRINGS, maxrings);

  if(!ioctl(ixj, PHONE_RING))
  {
    printf("Nem veszik fel.\n");
  }
  else
  {
  printf("Felvették a telefont.\n");
  }
  close(ixj);
}
```

3. lista A kivétel-adatszerkezet használata

```
void
getdata(int ixj1)
{
   fd_set              rfds,wfds,efds;
   struct timeval      tv;
   union telephony_exception ixje1;
   int                 nmax, size, state, nlen;
   char                buf[480], jbuf[480],
                       dtmf1;
   char                date[5], time[5], clen[2],
                       pnum[11], cname[80];

   nmax = ixj1+1;

   /* fájlleírók törlése */
   FD_ZERO(&rfds);
   FD_ZERO(&wfds);
   FD_ZERO(&efds);

   /* mindegyik beállítása az ixj1-re */
   FD_SET(ixj1, &rfds);
   FD_SET(ixj1, &wfds);
   FD_SET(ixj1, &efds);

   /* a select időtartamát állítsuk kicsire */
   tv.tv_sec = 0;
   tv.tv_usec = 300;

   /* várjuk az idő leteltére, vagy
      vagy eseményre a fájlleíróban */
   select(nmax, NULL, &wfds, &efds, &tv);

   if(FD_ISSET(ixj1,&wfds))
   {
      /* A fájlleíró írásra kész
         - küldjünk adatokat! */
   }

   if(FD_ISSET(ixj1,&rfds))
   {
      /* A fájlleíró olvasásra kész
         - olvassuk be az adatokat! */
   }

   if(FD_ISSET(ixj1,&efds))
   {
     /* megkérdezzük az eszközt,
         hogy milyen kivétel történt */
     ixje1.bytes = ioctl(ixj1,
                   PHONE_EXCEPTION);

     /* ellenőrizzük, hogy a felhasználó
        lerakta vagy felvette a telefont */
     if(ixje.bits.hookstate)
     {
       if(ioctl(ixj1, PHONE_HOOKSTATE))
       {
         printf("Felvette\n");
       }
       else
       printf("Lerakta\n");
     }
   }
}
```

kodek (tömörítetlen 16 bites hangminták) alapértelmezett adatkocka-mérete 240 bájt – ez 30 ezredmásodpercnyi adatnak felel meg 8000 Hz-en mintavételezve. Az eszközről minden alkalommal 240 bájtot lehet beolvasni, vagy semmit. Természetesen ez a viselkedés megváltoztatható. Különböző ioctl hívásokkal beállítható a tömörítést nem végző kodekekhez az adatkocka mérete.

A telefoneszközök vezérlése – az ioctl rendszerhívás

A telefoneszközzel közlendő parancsokat, amelyek egy bizonyos választ váltanak ki, nem a write hívással írjuk az eszközre, hiszen mint fent említettük, csak a hangadatok kerülnek írás műveleten keresztül az eszközre. Az alapvető telefonszolgáltatások vezérlését ioctl rendszerhívásokkal valósítják meg. Ezeket az alapvető ioctl függvényeket a /usr/include/linux/telephony.h fejlécfájlban adták meg. A gyártók kibővíthetik ezt az alapvető függvénykészletet, de azok a függvények csak a saját eszközmeghajtójukra korlátozódnak, és nem lesznek részei a közös linuxos telefonos API-nak.
Ezt a lehetőséget egy példával világíthatjuk meg a legjobban. A telefonos API-t egy Quicknet Internet PhoneJACK kártyával használjuk, amelyhez egy telefon csatlakozik (a telefonos szakemberek FXS kapunak vagy POTS kapunak nevezik). A *2. lista* rövid példaprogramja megcsörgeti a telefont.
Láthatjuk, hogy ha a felhasználó nem adja meg a parancssorban az eszköz nevét, akkor a /dev/phone0 nyílik meg. A csengetések legnagyobb számát egy ioctl hívással állítjuk be a PHONE_MAXRINGS állandó segítségével. Ezután a telefont csörgésre késztetjük a PHONE_RING ioctl által. A példaprogram egy egyszerűsített változata az LGPL-es ring.c modulnak, amely a Quicknet Software Developer Kit (SDK) része. Egyszerűsége miatt nem várhatunk tőle többet, mint hogy bemutassa a telefoneszköz használatának módját és az ioctl hívásokon keresztüli vezérlést, de a valódi programokat sem kell sokkal jobban bonyolítani, az API elég egyszerű. Az összes linuxos telefonos API-hoz tartozó ioctl állandó a /etc/include/linux/telephony.h fejlécfájlban van meghatározva.
A hangkártyákhoz írott programok könnyen átültethetők telefoneszközökre, ugyanis csak a hangadatokat írjuk és olvassuk a read és a write rendszerhívásokkal. Ráadásul az ioctl hívásokban használt alacsony szintű állandók úgy lettek meghatározva, hogy ne ütközzenek a létező hangkártyák ioctl állandóival. Egy hangkártyás alkalmazás átültetése telefonkártyára főként abból áll, hogy a saját kódunk által kiadott hangkártyás ioctl hívásokra adott hibákat kezeljük. Lehetségessé válik (bár talán nem könnyű) egy héjalkalmazást írni, amely megnyitja a telefoneszközt, elindít egy gyermekfolyamatot, (amely megörökli a telefoneszköz megnyitott fájlleíróját) és ebben a folyamatban futtatja a hangkártyával működő alkalmazást. Bár ez nem teljesen átlátszó, de lehetséges és a gyakorlatban nem is lehet olyan nehéz.

Az aszinkron eseményértesítő

Az eszköz telefonos oldalán bekövetkező eseményekről a telefont működtető felhasználói térben futó alkalmazást értesíteni kell.
A régi, nem túl szép módszer erre az, hogy a program állandóan lekérdezi az eszköz állapotát és a változásokat. A Linux telefonos

API-ja természetesen elkerüli ezt a megoldást, és két másik módszert alkalmaz, mindkettőt általában „aszinkron eseményértesítőnek” nevezik. Az első módszer jelzéseket (signal) használ, a második a kivételbitet állítja be a fájlleíró kivétel-adatszerkezetében. Tekintsük át sorban mindkét módszert.

A jelzések használata eseményértesítőként a következő három lépést foglalja magába: először SIGIO jelzéshez kell a jelzéskezelő függvényt előkészíteni és megadni, másodszor be kell állítani a futó folyamat folyamatazonosítóját (PID), hogy fogadja a jelzést, harmadszor engedélyezni kell a jelzés létrehozását a megnyitott fájlleírón az fcntl rendszerhívás segítségével. E három lépést részletesen tárgyalja *W. Richard Stevens* Advanced Programming in the UNIX Environment című könyvében a 12. fejezetben. (A könyvet egyébként is érdemes elolvasni.) Ismét egy rövid példával világíthatjuk meg a legjobban a dolgot. Tegyük fel, hogy van egy megnyitott fájlleírónk, a neve ixj1, és ez egy megnyitott telefoneszköz. A következő programrészlet engedélyezi a jelzésekkel megvalósított aszinkron eseményértesítést:

```
signal(SIGIO, &getdata);
fcntl(ixj1, F_SETOWN, getpid());
oflags1 = fcntl(ixj1, F_GETFL);
fcntl(ixj1, F_SETFL, oflags1 | FASYNC);
```

A kapcsolódó jelzéskezelő függvény (a getdata a fenti kódban) dolgozza fel az adatokat. Amikor jelzés érkezik, nem tudjuk, hogy milyen esemény történt, csak azt tudjuk, hogy történt valami. A programunknak ezután meg kell kérdeznie a telefoneszköztől egy ioctl híváson keresztül, hogy milyen eseményt észlelt. Ráadásul, ha egynél több megnyitott eszközfájlleíróval van dolgunk, nem tudhatjuk, hogy melyik küldte a jelzést. A jelzésekkel nehéz dolgozni, nem megbízhatóak többszálú környezetben, ezért egyes fejlesztők elkerülik őket. Ezen tényezők korlátozzák a módszer hatékonyságát, ez a hibalehetőség használhatóbb kiküszöbölése felé vezet: a fájlleírók kivétel-adatszerkezetében levő „kivételbit” használata felé.

A programok gyakran élnek azzal a lehetőséggel, hogy beállítják az írási és az olvasási adatszerkezeteket valamilyen értékre, azután a select( ) rendszerhívással kivárják, hogy a fájlleíró írható vagy olvasható legyen. Kevésbé ismert a select( ) rendszerhívás a kivétel-adatszerkezetre. A linuxos telefonos API ezt a kivétel-adatszerkezetet használja a telefonos események jelzésére. A *3. lista* egy egyszerű példát mutat az ixj1 fájlleíró felhasználására.

Ez a végtelenül leegyszerűsített (és nem túl hasznos) példa kizárólag arra alkalmas, hogy bemutassa a kivétel-adatszerkezet és a select( ) használatát az események észlelésére. Esemény bekövetkeztekor a telefoneszköz meghajtóprogramja beállítja a kivétel-adatszerkezet megfelelő bitjét, ennek hatására a select( ) visszatér. A felhasználó megvizsgálhatja az olvasási, az írási és a kivétel-adatszerkezeteket, és megállapíthatja, hogy az eszközmeghajtó által megjelölt saját fájlleírója milyen műveletek elvégzésére kész. Ha az adatok olvasásra készek, akkor az FD_ISSET(ixj1,&rfds) kifejezés IGAZ értékkel tér vissza. Az FD_ISSET(ixj1,&wfds) kifejezés akkor tér vissza IGAZ értékkel, ha az eszköz kész adatok fogadására. Végül az FD_ISSET(ixj1,&efds) akkor ad IGAZ értéket, ha telefonos esemény következett be. Hogy állapíthatjuk meg, hogy pontosan mi történt?

Az API egy különleges PHONE_EXCEPTION ioctl hívást tesz lehetővé, ehhez a visszatérési érték megfejtését segítő telephony_exception adatszerkezet tartozik. A hívás által az adatszerkezetben beállított bitekből következtethetünk arra, hogy milyen telefonos esemény történt (sokféle lehet). A fenti példában a „hookstate” bitet vizsgáltuk az if(ixje.bits.hookstate) kifejezéssel. A bit beállítása jelzi az állapotváltozást. Ezután egy ioctl hívással döntjük el, hogy letették vagy felvették a telefont. Ennek a módszernek a részletes magyarázata meghaladja cikkünk kereteit, de az érdeklődők utánanézhetnek a /usr/include/linux/telephony.h fájlban, hogy milyen események észlelhetők.

A jelenleg elérhető nyílt forráskódú programok

Már ma is sok nyílt forráskódú program használja ezt az API-t. A legjobban ismert és legszélesebb körben használt program az ophone, amely a nyílt forráskódú OpenH323 programkönyvtárat használó konzolos alkalmazás. Az ophone része az OpenH323 projektnek ➲ http://www.openh323.org, és több ezer ember használja nap mint nap ingyenes, csúcsminőségű, Interneten keresztüli telefonhívásokhoz. Az ophone nem csak a Linux telefonos API-ját támogatja teljes mértékben, hanem más H.323-alapú termékekkel is megfeleltethető, mint például a Microsoft NetMeeting vagy a Cisco hangátvitelre képes útválasztói. Ennél részletesebben terjedelmi okokból nem írhatunk erről a remek programról, de mindenkit bátorítunk a webhelyük megtekintésére. Az OpenH323 programkönyvtárat kifejlesztő céget nemrég vásárolta fel a Quicknet Technologies vállalat, hogy továbbra is biztosítva legyen ennek a nyílt forráskódú projektnek a fejlődése. Az ilyen üzleti háttérrel és a nyílt forráskód melletti elkötelezettséggel bíró OpenH323 projekt várhatóan még sikeresebb lesz a közeljövőben.

Összefoglalás

A Linux telefonos API-ja egységes és teljes felületet teremt a telefont használó programok Linux alatti fejlesztéséhez. Bár még csak egy gyártó (a Quicknet Technologies Inc.) szállít az API-nak teljesen megfelelő meghajtóprogramot, sokan mások is dolgoznak hasonló meghajtóprogramok fejlesztésén. Az API egyszerű felépítésű, gondosan megtervezett, nem ütközik a jelenlegi hangkártyák API-jával, és ugyanazzal a felülettel képes többféle gyártó termékét kiszolgálni. A következő évben biztosan kifejlesztenek még néhány nagyon érdekes telefonos programot Linuxra.

Greg Herlein (greg@herlein.com)
A californiai San Franciscóban él és dolgozik.1994 óta lelkes Linux-fejlesztő. Jelenleg a Herlein Engineering Linux/Unix tanácsadással foglalkozó cég munkatársa.

Kapcsolódó címek

OpenH323 projekt
➲ http://www.openh323.org/

OpenSwitch projekt
➲ http://www.openswitch.org/

Quicknet Linux termékek
➲ http://www.linuxjack.com/

LinuxTelephony.Org
➲ http://www.linuxtelephony.org/

Asterisk PBX projekt
➲ http://www.asteriskpbx.org/

Voxilla – Nyílt forráskódú telefonálás
➲ http://www.voxilla.org/

OpenGatekeeper projekt
➲ http://www.opengatekeeper.org/

Packetizer – a VoIP módszertani oldala
➲ http://www.packetizer.com/

Az IETF IP-telefonálás munkacsoportja
➲ http://www.ietf.org/html.charters/iptel-charter.html