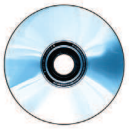


Bevezetés az ALSA alapú hangprogramozásba

A 2.6-os rendszermag hangkezelő alrendszerének minden szolgáltatását egyetlen teljes értékű API-n keresztül vehetjük igénybe.

Az *ALSA* az *Advanced Linux Sound Architecture* (fejlett linuxos hangkezelő alrendszer) rövidítése. Rendszermag-illesztőprogramokból, egy alkalmazásprogramozási felület (application programming interface, *API*) könyvtárból és *Linux* alatti hangkezelést támogató segédprogramokból áll. Írásomban szeretném röviden áttekinteni az *ALSA Projectet* és szoftveres összetevőit. Elsősorban az *ALSA PCM* felületének programozásával fogok foglalkozni, a próbálgatást, az ismerkedést pedig kódrészletek mellékelésével szeretném elősegíteni.
Az *ALSA* megismerése nemcsak azért lehet érdekes, mert újdonság, hanem mert nem ez az egyetlen hangkezelő *API*. Az *ALSA* kiváló választás, ha alacsony szintű hangkezelő eljárásokat szeretnénk megvalósítani, vagy különleges, a többi hangkezelő *API* által nem támogatott szolgáltatásokat szeretnénk használni. Ha meglévő hangkezelő alkalmazással rendelkezünk, dönthetünk úgy, hogy az *ALSA* illesztőprogramok natív támogatásával bővítjük. Ha a hangkezelés kevésbé érdekel bennünket, és például csak hangfájlokat szeretnénk lejátszani, akkor a magas szintű hangkezelő eszközkészletek valamelyike, mint az *SDL*, az *OpenAL* vagy a saját terjesztésünkben szereplő egyéb megoldás számunkra jobb választás lehet. Az *ALSA* használatakor gyakorlatilag az *ALSA* támogatására képes rendszermaggal ellátott rendszerekre korlátozzuk önmagunkat.

Az ALSA története

Az *ALSA Project* azért indult el, mert a *Linux* rendszermag hangkezelő illesztőprogramjai (az *OSS/Free* illesztőprogramok) aktív gondozás nélkül maradtak, így az újabb hangkezelő megoldásokat sem voltak képesek támogatni. A tervezetet *Jaroslav Kysela* indította, aki korábban egy hangkártya-illesztőprogramot készített. Idővel egyre több fejlesztő csatlakozott hozzá, egyre több hangkártya támogatása valósult meg, az *API* felépítése pedig fokról fokra finomodott.
A 2.5-os sorozat fejlesztése során az *ALSA* a *Linux* rendszermag hivatalos része lett. A 2.6-os rendszermag megjelenésével az *ALSA* az üzembiztos ág részévé vált, és várhatóan széles körben el fog terjedni.

A digitális hangkezelés alapjai

A hangok lényegében változó légnyomáshullámok, ezeket valamilyen átalakítóval, például mikrofonnal formáljuk elektromos jelekké. Az analóg-digitális átalakítók (*analog-to-digital converter*, *ADC*) az analóg feszültségeket diszkrét értékeknek feleltetik meg, ezeket mintáknak nevezzük. A műveletet rendszeres időközönként végzik el, amit mintavételi időnek hívunk. Ha a mintákat egy digitális-analóg átalakítóra küldjük, majd kimeneti átalakítóra, például egy hangszóróra továbbítjuk, akkor helyre tudjuk állítani az eredeti hangot.
A minták méretét bitekben szokták megadni, ez az egyik olyan tényező, amely meghatározza, hogy a digitális formátummal milyen pontosan tudjuk ábrázolni a hangot.
A hangminőség másik fontos befolyásoló eleme a mintavételi gyakoriság. A *Nyquist*-tétel kimondja, hogy a legmagasabb visszaadható frekvencia egyenlő a mintavételi frekvencia felével.

ALSA alapismeretek

Az *ALSA* különféle hangkártyákhoz készült illesztőprogramok gyűjteménye, valamint tartalmaz egy *API* könyvtárat is, ez a *libasound*. Az alkalmazásfejlesztők számára az *API* használata javasolt, és nem a rendszermag felületé.
A könyvtár magasabb szintű és a fejlesztők számára barátságosabb felületet biztosít, továbbá lehetővé teszi az eszközök logikai elnevezését, így a fejlesztőknek nem kell elveszniük az alacsony szintű részletekben, mint például az eszközfájlok kezelése.
Az *OSS/Free* illesztőprogramok programozása ezzel szemben rendszermag szintű hívásokkal történik, amihez a fejlesztőknek eszközfájlneveket kell megadniuk, illetve számos műveletet *ioctl* hívásokkal kell elvégezniük.
A visszirányú együttműködés lehetősége érdekében az *ALSA* olyan rendszermagmodulokat is tartalmaz, amelyek emulálják az *OSS/Free* hangkezelő illesztőprogramokat, így a legtöbb hangkezelő alkalmazás módosítások nélkül is tovább használható. Az *OSS/Free API*-t rendszermagmodulok nélkül a *libaoss* burkolókönyvtárral lehet emulálni.
Az *ALSA* a beépülő modulokat is támogatja, ezekkel új eszközök támogatása is megvalósítható, akár teljes egészében szoftveresen megvalósított képzetes eszközök létrehozására is mód nyílik. Az *ALSA* számos parancssori segédprogramot tartalmaz, egyaránt található köztük keverő, lejátszóprogram, valamint az egyes hangkártyák különleges képességeinek vezérlésére szolgáló kiegészítők.

Az ALSA felépítése

Az *ALSA API* az általa támogatott fontosabb felületek alapján osztható fel:

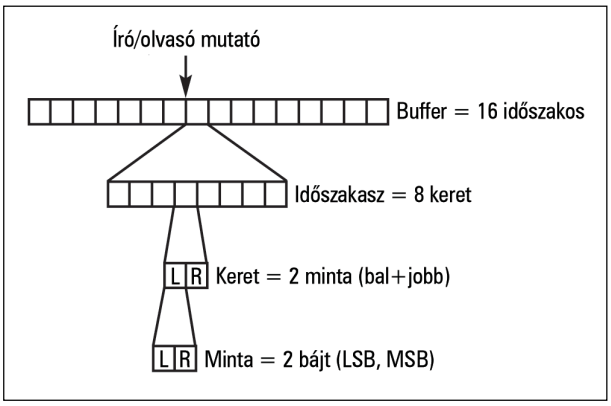
- *Vezérlőfelület*: Általános célú felület a hangkártyák regisztereinek kezelésére és az elérhető eszközök lekérdezésére.
- *PCM felület*: A digitális hangfelvételek és lejátszások kezelésére szolgáló felület. A továbbiakban erről a felületről lesz szó, ugyanis a digitális hangkezelő alkalmazásokban ez jut a legfontosabb szerephez.
- *Nyers MIDI felület*: Az elektronikus hangszerek terén elterjedt szabvány, a *MIDI* (*Musical Instrument Digital Interface*, digitális hangszer felület) támogatását valósítja meg. Ez az *API* a hangkártyán található *MIDI* busz elérését teszi lehetővé. A nyers interfész a *MIDI* eseményekkel közvetlenül dolgozik, a protokoll és az időzítések kezeléséért a programozó felelős.
- *Időzítő felület*: A hangesemények szinkronizálása céljából biztosítja a hangkártyákon található időzítő eszközök elérését.
- *Sorrendvezérlő felület*: A nyers *MIDI* felületnél magasabb szintű *MIDI* programozó és hangkeltő felület. A *MIDI* protokoll és az időzítések túlnyomó részét képes kezelni.
- *Keverő interfész*: A hangkártyákon lévő, a jeleket irányító és a hangerőszinteket szabályozó eszközöket vezérli. A vezérlő felületre épül.

Eszköznevek

A könyvtári *API* eszközfájlok helyett logikai eszköznevekkel dolgozik. Az eszköznevek valós, fizikai eszközök és beépülő modulok egyaránt lehetnek. A hardvereszközök a `hw:i,j` formátumot használják, ahol az `i` a kártya számát, a `j` pedig az adott kártyán lévő eszközt adja meg. Az első hangeszköz a `hw:0,0`. Az írásomhoz tartozó példákban szereplő `default` (alapértelmezett) álnév az első hangkezelő eszközre hivatkozik. A beépülő modulok egyéb egyedi neveket használnak, a `plughw:` például olyan beépülő modul, amely hardvereszközhöz biztosít hozzáférést, de egyéb, a hardvereszköz által alapesetben nem támogatott szolgáltatásokat is biztosít, például a mintavételi gyakoriság megváltoztatását. A `dmix` és a `dshare` beépülő modul segítségével több adatfolyamot tudunk egyesíteni, illetve egyetlen folyamot dinamikusan tudunk megosztani különböző alkalmazások közt.

Hangpufferek és adatátvitel

A hangkártyák rendelkeznek egy pufferrel, amely tárolja a rögzített mintákat. Ha a pufferben elegendő minta gyűlt fel, megszakítást hoz létre. A rendszermag hangkezelő illesztőprogramja közvetlen memória-hozzáférést (*direct memory access*, *DMA*) használ a minták továbbítására a megfelelő alkalmazás memóriabeli pufferébe. A lejátszás hasonlóan folyik, *DMA* használatával az alkalmazás memóriabeli pufferének tartalma a hangkártya hardveres pufferébe továbbítódik.
A hardverpufferek körkörös elven működnek, vagyis végük elérésekor az újabb adatok beírása az elejükre történik. A hardver- és az alkalmazáspufferben egyaránt mutató jelzi az aktuális írási-olvasási helyzetet. A rendszermagon kívüli összetevők közül csak az alkalmazáspuffer érdemes figyelemre, ezért a továbbiakban csak ezzel foglalkozunk.



1. ábra Az alkalmazáspuffer

A puffer méretét *ALSA* könyvtárhívásokkal lehet beállítani. A puffer akár egészen nagy méretű is lehet, ám ekkor tartalmának egyetlen műveletben való továbbítása elfogadhatatlan mértékű késleltetéshez, lappangási időhöz vezet. Ennek elkerülése érdekében az *ALSA* a puffert időszak sorozatokra osztja (ezeket *OSS/Free* alatt töredékeknek hívják), és az adatokat időszak egységekben továbbítja.
Egy időszak kereteket tartalmaz, ezek mindegyike adott időpontban rögzített mintákat foglal magába. Egy sztereó eszköznél egy-egy keretnek két csatornához kell mintákat tartalmaznia. Az *1. ábrán* – elméleti értékeket használva – a pufferek időszakokra, keretekre és mintákra osztását szemléltettem. Ebben az esetben a bal és a jobb oldali csatorna adatai felváltva szerepelnek a keretekben; ezt átlapolt módnak nevezzük. A nem átlapolt mód támogatása – ennél az egyik csatorna adatai a másik csatorna adatai után szerepelnek – szintén megoldott.

Alul- és túlcsordulás

Ha egy hangeszköz aktív, akkor folyamatos adattovábbítás folyik a hardver- és az alkalmazáspuffer között. Adatrögzítéskor (felvételkor), ha az alkalmazás nem olvassa ki elég gyorsan a puffert, akkor annak tartalma a körkörös működés miatt felülírásra kerül az új adatokkal. Az ilyen típusú adatvesztést túlcsordulásnak nevezzük. Ha lejátszáskor az alkalmazás képtelen kellő gyorsasággal biztosítani a szükséges adatokat, a puffer kiürül, ami alulcsordulási hibát eredményez. Az *ALSA* leírásában előfordul, hogy mindkét hibára az `XRUN` kifejezéssel hivatkoznak. A helyesen tervezett alkalmazások az `XRUN` hibák számát a lehető legkisebbre szorítják le, és fellépésük után is képesek folytatni működésüket.

Egy átlagos hangkezelő alkalmazás

A *PCM* felületet használó programok általában az alábbi lépéseket követik:

- a felület megnyitása rögzítés vagy lejátszás céljából
- a hardver beállításainak megadása
- (hozzáférési mód, adatformátum, csatornák, gyakoriság stb.)

Az adatok feldolgozása közben:

- PCM adatok olvasása (felvétel)
- vagy PCM adatok írása (lejátszás)
- interfész lezárása

1. kódrészlet Néhány PCM típus és formátum megjelenítése

```
#include <alsa/asoundlib.h>

int main() {
  int val;

  printf("Az ALSA könyvtár változatszáma: %s\n",
         SND_LIB_VERSION_STR);

  printf("\nPCM adatfolyam típusok:\n");
  for (val = 0; val <= SND_PCM_STREAM_LAST; val++)
    printf("  %s\n",
      snd_pcm_stream_name((snd_pcm_stream_t)val));

  printf("\nPCM hozzáférési típusok:\n");
  for (val = 0; val <= SND_PCM_ACCESS_LAST; val++)
    printf("  %s\n",
      snd_pcm_access_name((snd_pcm_access_t)val));

  printf("\nPCM formátumok:\n");
  for (val = 0; val <= SND_PCM_FORMAT_LAST; val++)
    if (snd_pcm_format_name((snd_pcm_format_t)val)
      != NULL)
      printf("  %s (%s)\n",
        snd_pcm_format_name((snd_pcm_format_t)val),
        snd_pcm_format_description(
                          (snd_pcm_format_t)val));

  printf("\nPCM alformátumok:\n");
  for (val = 0; val <= SND_PCM_SUBFORMAT_LAST;
       val++)
    printf("  %s (%s)\n",
      snd_pcm_subformat_name((
        snd_pcm_subformat_t)val),
      snd_pcm_subformat_description((
        snd_pcm_subformat_t)val));

  printf("\nPCM állapotok:\n");
  for (val = 0; val <= SND_PCM_STATE_LAST; val++)
    printf("  %s\n",
      ➥snd_pcm_state_name((snd_pcm_state_t)val));

  return 0;
}
```

Az alábbiakban ténylegesen működő kódrészleteket is látni fogunk. Mindenkinek javasolom, hogy fordítsa és futtassa le ezeket saját ***Linux*** rendszerén, vizsgálja meg a kimenetet, és próbálkozzon meg a javasolt módosításokkal. A cikkhez tartozó példaprogramok teljes változata a 65. CD Magazin/ALSA könyvtárban megtalálható.

Az *1. kódrészlet* az ***ALSA*** által használt ***PCM*** adattípusok és beállítási értékek egy részét jeleníti meg. Az első követelmény a fejrészfájl beillesztése, ez tartalmazza az ***ALSA*** könyvtári függvényeinek megadásait. A megjelenő megadások egyike az ***ALSA*** változatszáma.

A program fennmaradó része különféle ***PCM*** adattípusokon lépeget végig, kezdve az adatfolyam típusokkal. Az ***ALSA*** az utoljára számba vett értékhez szimbolikus neveket is szolgáltat, illetve egy segédfüggvénnyel egy karakterláncban rövid leírást is képes mellékelni az egyes értékekhez. Amint a kimenetből is látható, az ***ALSA*** rengeteg, a saját rendszeremen például 38 különféle formátum támogatására képes.

A programot fordításkor előbb össze kell kapcsolni az ***ALSA libasound*** könyvtárával, ezt az esetek többségében a `-lasound` kapcsolóval tehetjük meg. Az ***ALSA*** könyvtári függvények egy része használja a `dlopen` függvényt, illetve lebegőpontos műveleteket végez, ezért a `-ldl` és a `-lm` kapcsolóra úgyszintén szükség lesz.

A *2. kódrészlet* az alapértelmezett ***PCM*** eszközt nyitja meg, megad néhány beállítást, majd a megjeleníti a hardveres beállítások túlnyomó részének értékét. Hangrögzítést vagy -lejátszást nem végez. Az `snd_pcm_open` hívással megtörténik az alapértelmezett ***PCM*** eszköz megnyitása, illetve a hozzáférési mód beállítása `PLAYBACK`-re, vagyis lejátszásra. A függvény első átadott értékében egy kezelőt (***handle***) ad vissza, a későbbi hívásokban ezzel történik a ***PCM*** adatfolyam manipulálása. A legtöbb ***ALSA*** könyvtárhíváshoz hasonlóan a függvény egy egész állapotértéket ad vissza, ha ez negatív, az hibára utal. Ebben az esetben ellenőrizzük a visszatérési értéket, az `snd_strerror` függvénnyel megjelenítjük a hibaüzenetet, végül kilépünk. A könnyebb áttekinthetőség érdekében a hibaellenőrzések túlnyomó részét kihagytam a példaprogramokból. Valódi alkalmazásban természetesen minden ***API*** hívás visszatérési értékét ellenőrizni kell, és gondoskodni kell a megfelelő hibakezelésről.

Ahhoz, hogy meg tudjuk adni az adatfolyam hardveres beállításait, kell foglalnunk egy `snd_pcm_hw_params_t` típusú változót; ezt az `snd_pcm_hw_params_alloca` makróval tehetjük meg. Következő lépésként az `snd_pcm_hw_params_any` függvénnyel elvégezzük a változó kezdeti értékadását, ekkor az imént megnyitott ***PCM*** adatfolyamot is átadjuk.

Most ***API*** hívásokkal megadjuk a kívánt hardverbeállításokat, ehhez felhasználjuk a ***PCM*** folyamkezelőt és a hardverbeállításokat tartalmazó adatszerkezetet. Az adatfolyamot átlapolt módra állítjuk, továbbá 16 bites mintaméretet, kétcsatornás működést és 44100 b/s mintavételi gyakoriságot választunk. A mintavételi gyakoriságnál ügyeljünk arra, hogy a hangeszköz nem biztos, hogy minden mintavételi gyakoriságot megfelelően támogat. A kívánt értékhez legközelebb álló és támogatott mintavételi gyakoriságot az `snd_pcm_hw_params_set_rate_near` függvénnyel tudjuk kiválasztani. A hardverbeállítások addig nem lépnek érvénybe, amíg az `snd_pcm_hw_params` függvényt meg nem hívjuk.

A program fennmaradó része különféle a ***PCM*** adatfolyamra vonatkozó beállításokat jelenít meg, köztük az időszak hosszát és a puffermétereket. A megjelenő eredmények a hangeszköztől nagymértékben függenek.

Miután lefuttattuk a programot saját gépünkön, próbáljunk kísérletezni egy kicsit, és végezzünk el néhány módosítást. Változtassuk az eszköznevet `default`-ról `hw:0,0`-ra vagy `plughw:`-re, és vizsgáljuk meg a módosítás eredményét. Módosítsuk a hardverbeállításokat is, és vizsgáljuk meg, hogy a megjelenő értékek miben térnek el.

A *3. kódrészlet* az előbbi példa kibővítése, ebben hangmintákat is írunk a hangkártyára, vagyis hangokat szólaltatunk

2. kódrészlet PCM eszköz megnyitása és a beállítások megadása

```
/*

A példaprogram az alapértelmezett PCM eszközt
nyitja meg, megad néhány beállítást, majd a meg-
jeleníti a hardveres beállítások túlnyomó része-
nek értékét. Hangrögzítést vagy -lejátszást nem
végez.

*/

/* Az újabb ALSA API használata*/

#define ALSA_PCM_NEW_HW_PARAMS_API

/* Az ALSA könyvtári API megadása teljes
egészében ebben a fejrészben található
*/

#include <alsa/asoundlib.h>

int main() {
  int rc;
  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  unsigned int val, val2;
  int dir;
  snd_pcm_uframes_t frames;

  /* PCM eszköz megnyitása lejátszásra */

  rc = snd_pcm_open(&handle, “default”,
                    SND_PCM_STREAM_PLAYBACK, 0);
  if (rc < 0) {
    fprintf(stderr,
            “A PCM eszközt nem lehet megnyitni:
             ➥%s\n”, snd_strerror(rc));
    exit(1);
  }

  /* A hardverbeállításokat tároló objektum
     lefoglalása */

  snd_pcm_hw_params_alloca(&params);

  /* Feltöltése alapértékekkel */

  snd_pcm_hw_params_any(handle, params);

  /* A kívánt hardverbeállítások megadása */

  /* Átlapolt mód */

  snd_pcm_hw_params_set_access(handle, params,
    ➥SND_PCM_ACCESS_RW_INTERLEAVED);

  /* Előjeles, 16 bites, kis indián formátum */

  snd_pcm_hw_params_set_format(handle, params,
    ➥SND_PCM_FORMAT_S16_LE);

/* Két csatorna (sztereó) */

  snd_pcm_hw_params_set_channels(handle, params,
  ➥2);

  /* 44100 bit/másodperc mintavételi gyakoriság
     (CD minőség) */

  val = 44100;
  snd_pcm_hw_params_set_rate_near(handle,
    ➥params, &val, &dir);

  /* A beállítások kiírása az illesztőprogram
     felé */

  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr, “A hardverbeállításokat
                  ➥nem sikerült megadni: %s\n”,
                  ➥snd_strerror(rc));
    exit(1);
  }

  /* A PCM felület adatainak megjelenítése */

  printf(“PCM kezelő neve = ´%s´\n”,
         ➥snd_pcm_name(handle));

  printf(“PCM állapot = %s\n”,
    ➥snd_pcm_state_name(snd_pcm_state(handle)));
  snd_pcm_hw_params_get_access(params,
    ➥(snd_pcm_access_t *) &val);

  printf(“Hozzáférés típusa = %s\n”,
   ➥snd_pcm_access_name((snd_pcm_access_t)val));

  snd_pcm_hw_params_get_format(params, &val);
  printf(“Formátum = ´%s´ (%s)\n”,
   ➥snd_pcm_format_name((snd_pcm_format_t)val),
   ➥snd_pcm_format_description(
   ➥(snd_pcm_format_t)val));

  snd_pcm_hw_params_get_subformat(params,
  ➥(snd_pcm_subformat_t *)&val);
  printf(“Alformátum = ´%s´ (%s)\n”,
   ➥snd_pcm_subformat_name
   ➥((snd_pcm_subformat_t)val),
   ➥snd_pcm_subformat_description(
   ➥(snd_pcm_subformat_t)val));

  snd_pcm_hw_params_get_channels(params, &val);
```

```
printf("Csatornák = %d\n", val);
snd_pcm_hw_params_get_rate(params, &val, &dir);
printf("Gyakoriság = %d bps\n", val);

snd_pcm_hw_params_get_period_time(params,
➥&val, &dir);
printf("Időszakhossz = %d us\n", val);

snd_pcm_hw_params_get_period_size(params,
➥&frames, &dir);
printf("Időszakméret = %d frames\n",
        ➥(int)frames);

snd_pcm_hw_params_get_buffer_time(params,
➥&val, &dir);
printf("Pufferidő = %d us\n", val);

snd_pcm_hw_params_get_buffer_size(params,
➥(snd_pcm_uframes_t *) &val);
printf("Pufferméret = %d frames\n", val);

snd_pcm_hw_params_get_periods(params, &val,
➥&dir);
printf("Időszak/puffer = %d frames\n", val);

snd_pcm_hw_params_get_rate_numden(params,
➥&val, &val2);
printf("Pontos gyakoriság = %d/%d bps\n", val,
➥val2);

val = snd_pcm_hw_params_get_sbits(params);
printf("Számottevő bitek = %d\n", val);

snd_pcm_hw_params_get_tick_time(params,
➥&val, &dir);
printf("Ütésidő = %d us\n", val);

val = snd_pcm_hw_params_is_batch(params);
printf("Kötegelés = %d\n", val);

  val = snd_pcm_hw_params_is_block_transfer
  ➥(params);
  printf("Blokkos átvitel = %d\n", val);

  val = snd_pcm_hw_params_is_double(params);
  printf("Double érték = %d\n", val);

  val = snd_pcm_hw_params_is_half_duplex(params);
  printf("Váltakozó kétirányú = %d\n", val);

  val = snd_pcm_hw_params_is_joint_duplex
  ➥(params);
  printf("Csatolt kétirányú = %d\n", val);

  val = snd_pcm_hw_params_can_overrange(params);
  printf("Tartomány túllépési lehetőség = %d\n",
  ➥val);

  val =
  ➥snd_pcm_hw_params_can_mmap_sample_resolution
  ➥(params);
  printf("mmap lehetőség = %d\n", val);

  val = snd_pcm_hw_params_can_pause(params);
  printf("Szüneteltetési lehetőség = %d\n", val);

  val = snd_pcm_hw_params_can_resume(params);
  printf("Folytatási lehetőség = %d\n", val);

  val = snd_pcm_hw_params_can_sync_start(params);
  printf("Szinkron indítási lehetőség = %d\n",
  ➥val);

  snd_pcm_close(handle);

  return 0;
}
```

meg. A szabványos bemenetről összegyűjtünk annyi bájtot, amennyi egy időszakra elegendő, majd öt másodpercnyi adatot írunk ki a kártya felé.

A program eleje az előzővel megegyező – először megnyitjuk a *PCM* eszközt, majd megadjuk a hardverbeállításokat. Az *ALSA* által kiválasztott időszakhosszt használjuk, és ennek megfelelően állítjuk be a mintákat tároló puffer méretét is. Ezután meghatározzuk az időszakhosszt, amiből tudjuk, hogy a programnak hány időszakot kell feldolgoznia ahhoz, hogy öt másodpercnyi hanganyag álljon elő.

Az adatokat kezelő hurokban a szabványos bemenetről olvasunk, a puffert pedig egy időszaknyi mintával töltjük fel. Ha fájl végéről olvasunk, esetleg a kapott bájtok száma a várttól eltérő, akkor hibakezelést végzünk.

A *PCM* eszközre az adatokat az `snd_pcm_writei` hívással írjuk ki. Ennek működése a rendszermag `write` rendszerhívásához nagyon hasonló, kivéve azt, hogy a méret keretekben van megadva. Ellenőrizzük a visszatérési értéket, ebből különféle hibákra tudunk következtetni. Az `EPIPE` visszatérési érték alulcsordulásra utal, amelynek hatására a *PCM* adatfolyam `XRUN` állapotba kerül, az adatfeldolgozás pedig leáll. Ebből az állapotból normál esetben az `snd_pcm_prepare` függvény meghívásával billenthetjük ki a folyamot, amely ilyenkor `PREPARED` (előkészített) állapotba kerül. Ezután a következő alkalommal újra írhatunk adatokat a folyamba. Ha egyéb hibajelzést kapunk, akkor kijelezzük a kódját, majd továbblépünk. Végül, ha a kiírt keretek száma nem egyezik meg az elvárttal, szintén hibaüzenetet jelenítünk meg.

A program addig ismétli a műveletet, amíg öt másodpercnyi keret össze nem gyűlik, el nem éri a fájl végét vagy olvasási hiba nem történik a bemeneten. Ekkor kerül sor az `snd_pcm_drain` meghívására, amivel engedélyezzük a függőben lévő hangminták továbbítását, illetve lezárjuk a folyamot. Felszabadítjuk a dinamikusan foglalt puffert, majd kilépünk.

Bizonyára sokaknak feltűnt, hogy a program nagyjából működésképtelen, amíg a bemenetet át nem irányítjuk a kon-

3. kódrészlet Egyszerű hanglejátszás

```
/*

A példaprogram a szabványos bemenetről olvas,
majd öt másodpercnyi adatot ír az alapértelmezett
PCM eszközre.

*/

/* Az újabb ALSA API használata*/
#define ALSA_PCM_NEW_HW_PARAMS_API

#include <alsa/asoundlib.h>

int main() {
  long loops;
  int rc;
  int size;
  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  unsigned int val;
  int dir;
  snd_pcm_uframes_t frames;
  char *buffer;

  /* PCM eszköz megnyitása lejátszásra */
  rc = snd_pcm_open(&handle, "default",
  ➥SND_PCM_STREAM_PLAYBACK, 0);
  if (rc < 0) {
    fprintf(stderr,
           ➥"A PCM eszközt nem lehet megnyitni:
           ➥%s\n", snd_strerror(rc));
    exit(1);
  }

  /* A hardverbeállításokat tároló objektum
     lefoglalása */

  snd_pcm_hw_params_alloca(&params);

  /* Feltöltése alapértékekkel */
  snd_pcm_hw_params_any(handle, params);

  /* A kívánt hardverbeállítások megadása */

  /* Átlapolt mód */
  snd_pcm_hw_params_set_access(handle, params,
  ➥SND_PCM_ACCESS_RW_INTERLEAVED);

  /* Előjeles, 16 bites, kis indián formátum */
  snd_pcm_hw_params_set_format(handle, params,
  ➥SND_PCM_FORMAT_S16_LE);

  /* Két csatorna (sztereó) */
  snd_pcm_hw_params_set_channels(handle, params,
  ➥2);

  /* 44100 bit/másodperc mintavételi gyakoriság
     (CD minőség) */

  val = 44100;
  snd_pcm_hw_params_set_rate_near(handle, params,
  ➥&val, &dir);

  /* Az időszakméret beállítása 32 keretre */
  frames = 32;
  snd_pcm_hw_params_set_period_size_near(handle,
  ➥params, &frames, &dir);

  /* A beállítások kiírása az illesztőprogram
     felé */

  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr,
           ➥"A hardverbeállításokat nem sikerült
           ➥megadni: %s\n",
           ➥snd_strerror(rc));
    exit(1);
  }

  /* Kellően nagy méretű, egy időszakot tárolni
     képes puffert használunk. */

  snd_pcm_hw_params_get_period_size(params,
  ➥&frames, &dir);

  /* 2 bájt/minta, 2 csatorna */
  size = frames * 4;
  buffer = (char *) malloc(size);

  /* A hurkot öt másodpercig futtatjuk */
  snd_pcm_hw_params_get_period_time(params,
  ➥&val, &dir);

  /* 5 másodperc mikroszekundumokban mérve,
     osztva az időszakkal */
  loops = 5000000 / val;

  while (loops > 0) {
    loops--;
    rc = read(0, buffer, size);
    if (rc == 0) {
      fprintf(stderr, "Bemeneti fájl vége\n");
      break;
    } else if (rc != size) {
      fprintf(stderr,
             ➥"rövid bemenet: %d bájtot sikerült
             ➥olvasni\n", rc);
    }
    rc = snd_pcm_writei(handle, buffer, frames);
    if (rc == -EPIPE) {
      /* Az EPIPE alulcsordulást jelez. */
      fprintf(stderr, "Alulcsordulás történt\n");
      snd_pcm_prepare(handle);
```

```
    } else if (rc < 0) {
      fprintf(stderr,
              "writei hiba: %s\n",
              snd_strerror(rc));
    }  else if (rc != (int)frames) {
      fprintf(stderr,
              ➥"Rövid írás, %d keretet sikerült
              ➥kiírni\n", rc);
    }
  }

  snd_pcm_drain(handle);
  snd_pcm_close(handle);
  free(buffer);

  return 0;
}
```

4. kódrészlet Hangfelvétel

```
/*

A példaprogram az alapértelmezett PCM eszközről
olvas, majd öt másodpercnyi adatot ír a szabvá-
nyos kimenetre.

*/

/* Az újabb ALSA API használata*/
#define ALSA_PCM_NEW_HW_PARAMS_API

#include <alsa/asoundlib.h>

int main() {
  long loops;
  int rc;
  int size;
  snd_pcm_t *handle;
  snd_pcm_hw_params_t *params;
  unsigned int val;
  int dir;
  snd_pcm_uframes_t frames;
  char *buffer;

  /* PCM eszköz megnyitása felvételre */
  rc = snd_pcm_open(&handle, "default",
  ➥SND_PCM_STREAM_CAPTURE, 0);
  if (rc < 0) {
    fprintf(stderr,
            ➥"A PCM eszközt nem lehet megnyitni:
            ➥%s\n",  snd_strerror(rc));
    exit(1);
  }

  /* A hardverbeállításokat tároló objektum
    lefoglalása */
  snd_pcm_hw_params_alloca(&params);

  /* Feltöltése alapértékekkel */
  snd_pcm_hw_params_any(handle, params);

  /* A kívánt hardverbeállítások megadása */

  /* Átlapolt mód */
  snd_pcm_hw_params_set_access(handle, params,
  ➥SND_PCM_ACCESS_RW_INTERLEAVED);

  /* Előjeles, 16 bites, kis indián formátum */
  snd_pcm_hw_params_set_format(handle, params,
  ➥SND_PCM_FORMAT_S16_LE);

  /* Két csatorna (sztereó) */
  snd_pcm_hw_params_set_channels(handle, params,
  ➥2);

  /* 44100 bit/másodperc mintavételi gyakoriság
     (CD minőség) */
  val = 44100;
  snd_pcm_hw_params_set_rate_near(handle, params,
  ➥&val, &dir);

  /* Az időszakméret beállítása 32 keretre */
  frames = 32;
  snd_pcm_hw_params_set_period_size_near(handle,
  ➥params, &frames, &dir);

  /* A beállítások kiírása az illesztőprogram
     felé */
  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr,
            ➥"A hardverbeállításokat nem sikerült
            ➥megadni: %s\n",
            ➥snd_strerror(rc));
    exit(1);
  }

  /* Kellően nagy méretű, egy időszakot tárolni
     képes puffert használunk. */
  snd_pcm_hw_params_get_period_size(params,
  ➥&frames, &dir);

  /* 2 bájt/minta, 2 csatorna */
  size = frames * 4;
  buffer = (char *) malloc(size);

  /* A hurkot öt másodpercig futtatjuk */
  snd_pcm_hw_params_get_period_time(params,
  ➥&val, &dir);
  loops = 5000000 / val;
```

```
while (loops > 0) {
  loops--;
  rc = snd_pcm_readi(handle, buffer, frames);
  if (rc == -EPIPE) {
    /* Az EPIPE túlcsordulást jelez. */
    fprintf(stderr, "Túlcsordulás történt\n");
    snd_pcm_prepare(handle);
  } else if (rc < 0) {
    fprintf(stderr,
            "read hiba: %s\n",
            snd_strerror(rc));
  }  else if (rc != (int)frames) {
    fprintf(stderr, "Rövid olvasás, %d keretet
                    sikerült beolvasni\n", rc);
  }
  rc = write(1, buffer, size);
  if (rc != size)
    fprintf(stderr,
            "Rövid írás: %d bájtot sikerült
            kiírni\n", rc);
}

snd_pcm_drain(handle);
snd_pcm_close(handle);
free(buffer);

return 0;
}
```

zolról. Megpróbálkozhatunk például a véletlenszerű adatokat előállító */dev/urandom* eszközzel:

```
./pelda3 < /dev/urandom
```

A véletlenszerű adatokkal öt másodpercnyi fehérzajt tudunk előállítani. Következő lépésként megpróbálhatjuk a */dev/null*-t vagy */dev/zero*-t használni bemenetként, majd összehasonlíthatjuk az eredményt. Változtassunk meg néhány beállítást, például a mintavételi időt vagy az adatformátumot, és figyeljük a hatást.

A *4. kódrészlet* sokban hasonlít a harmadikhoz, ám itt ***PCM*** alapú felvételt végzünk. A ***PCM*** adatfolyam megnyitásakor `SND_PCM_STREAM_CAPTURE` üzemmódot állítunk be. A fő adatfeldolgozó hurokban az `snd_pcm_readi` függvénnyel olvassuk be a mintákat a hangeszközről, majd `write` hívásokkal juttatjuk a szabványos kimenetre. Eközben ellenőrizzük, hogy nincs-e alulcsordulás, amit szükség esetén a 3. kódrészletben látott módszerrel kezelünk.

A 4. kódrészletet futtatva nagyjából öt másodpercnyi adatot tudunk összegyűjteni, illetve a szabványos kimenetre küldeni, amit természetesen fájlba is írhatunk. Ha hangkártyánkhoz mikrofon is csatlakozik, akkor keverőprogram segítségével kiválaszthatjuk azt a felvétel forrásaként, illetve beállíthatjuk a felvételi szintet. Ha gondoljuk, elindíthatunk valamilyen CD-lejátszó programot is, és a felvételi forrást a CD-re állíthatjuk. Próbáljuk meg futtatni a 4. kódrészletet, a kimenetet pedig fájlba irányítani. Ezután a 3. kódrészlettel visszajátszhatjuk, amit rögzítettünk:

```
./pelda4 > hang.raw
./pelda3 < hang.raw
```

Ha hangkártyánk támogatja a teljes kétirányú működést, akkor a programokat csővezetékkel egymáshoz is csatolhatjuk, és a hangkártyáról érkező anyagot a következő paranccsal hallgatjuk meg: `./pelda4 | ./pelda3`. A ***PCM*** beállítások megváltoztatásával kipróbálhatjuk, hogy a mintavételi gyakoriság és a formátum módosítása milyen hatással jár.

Különleges lehetőségek

A példaprogramokban a ***PCM*** folyamokat blokkoló módban nyitottuk meg, vagyis a hívások addig nem tértek vissza, amíg az adatátvitel le nem játszódott. Interaktív, eseményvezérelt alkalmazásnál ezzel az alkalmazás futásának elfogadhatatlanul hosszú időkre való megakadását okozhatjuk. Az ***ALSA*** lehetővé teszi, hogy az adatfolyamokat nem blokkoló módban nyissuk meg, ilyenkor az olvasási és írási hívások azonnal visszatérnek. Ha az adatátvitelek függőben maradnak, és a hívások befejezésére nincs lehetőség, az ***ALSA*** az `EBUSY` foglaltságot jelző hibakódot adja vissza. Sok grafikus alkalmazás visszahívókkal kezeli az eseményeket. Az ***ALSA*** támogatja a ***PCM*** adatfolyamok aszinkron módban való megnyitását is, így mód nyílik arra, hogy adott mennyiségű mintaadat összegyűjtése után egy visszahívót futtassunk.

Az `snd_pcm_readi` és az `snd_pcm_writei` hívások használata a linuxos `read` és `write` rendszerhívásához hasonló. Az `i` betű arra utal, hogy a keretek átlapoltak (***interleaved***). Hasonló függvények nem átlapolt módhoz is léteznek. Linux alatt sok eszköz támogatja az `mmap` rendszerhívást, amelynek segítségével memóriabeli területeknek lehet megfeleltetni és mutatókkal lehet kezelni őket. Végül megemlíteném, hogy az ***ALSA*** támogatja a ***PCM*** csatornák `mmap` módban való megnyitását is, amivel hatékony, másolásmentes hozzáférést biztosít a hangadatokhoz.

Összefoglalás

Remélem, hogy írásommal sokaknak sikerült kedvet szereznem az ***ALSA*** kipróbálásához. Ahogy a terjesztések egyre inkább a 2.6-os rendszermag használatára térnek át, az ***ALSA*** egyre szélesebb körben fog elterjedni, fejlett szolgáltatásainak köszönhetően egyre jobb linuxos hangkezelő megoldások jelenhetnek majd meg.

Szeretnék köszönetet mondani ***Jaroslav Kyselának*** és ***Takashi Iwainak***, akik írásom vázlatának áttekintésével és tanácsaikkal rengeteget segítettek nekem.

Linux Journal 2004. október, 126. szám

Jeff Tranter 1992 óta használja, fejleszti és tárgyalja írásaiban a Linuxot. A kanadai Ottawában található Xandros Corporation munkatársa.