SDL – Multimédiás programozói könyvtár (3. rész)

Az előző alkalmak során eseménykezelésből is kaptunk egy kis ízelítőt. Most egy kicsit a fülünknek fogunk kedvezni, kezelésbe vesszük az SDL audio szolgáltatásait, majd az optikai meghajtónkat is felbolygatjuk. A grafika, multimédia, játékfejlesztés nélkülözhetetlen eszköze az időzítő. Erről is fogunk a cikk során tárgyalni.

Az *SDL* igen sok oldalú programozói eszköz. Véleményem szerint a legrövidebb út, ahhoz hogy *Linux* alatt könnyen tudjunk grafikát programozni. Az eddigi ismereteink alapján már nem lehet gondunk az eseményekkel és a videó szolgáltatással. Ideje, hogy belevágjunk az audio szolgáltatás alapjaiba, hiszen végül is egy multimédiás szolgáltatáscsaláddal van dolgunk, amiből nem maradhat ki a „fülbevaló” sem.

Indulhat a koncert

Kép van hang nincs. Mondhatnánk az eddig elhangzottak után, de ne legyünk türelmetlenek! A következőkben az *SDL* audio támogatása kerül előtérbe. Valójában most nem is az *SDL API*-ba tartozó dolgok következnek, az úgynevezett standard *SDL* könyvtárak egyikét fogjuk kicsit szemügyre venni, abból a célból, hogy végre elérjük, hogy *SDL* segítségével programozni tudjuk a hangeszközünket.
Egy egyszerű példát fogunk látni az *SDL_mixer* könyvtár használatára. Itt hívnám fel a figyelmet, hogy a fordítási direktívák között fel kell tüntetnünk majd, hogy az *SDL_mixer* könyvtárat szeretnénk használni a programunkhoz. Tehát egy *SDL_mixer*-es program fordítása így nézhet ki valahogy:

```
g++ `sdl-config --libs
➥--cflags` -lSDL_mixer
➥-pedantic -Wall -ansi
➥main.cpp -o sdl01
```

Ami az eddigiekhez képest változott az a `-lSDL_mixer` felbukkanása.
A kódunk is alakul picit, az eddigiekhez képest. Mivel a most következő eszközök nem tartoznak az *SDL API*-ba ezért nem lesz elég, hogy csak az *SDL.h*-t építsük be a programba, hanem szükségünk lesz az `SDL_mixer.h`-ra is.
Egy általános menete az *SDL*-ben egy hang lejátszásának a következő: hangeszköz megnyitása, hangfájl betöltése a memóriába, lejátszás, majd a hangadatok kitörlése a memóriából ha már nincs rá szükségünk. Lássuk hát a példánkat (*1. lista*).
A *1. listát* tekintve nézzük kicsit részletesebben az egészet, hiszen a kommentárok nem sok mindent

1. Lista Példaprogram hangeszköz használatára SDL segítségével

```
#include <iostream>
#include "SDL.h"
#include "SDL_mixer.h"

int main()
{
   // ez a mutató fogja mutatni a hangadatok
   // helyét a memóriában
    Mix_Music *HANGANYAG = NULL;

   // nyitunk egy ablakot, hogy az
   // eseménykezelést le tudjuk bonyolítani
    SDL_Surface *kepernyo;
    SDL_Event esemeny;
    // a formátumra vonatkozó beállítások
    // eltárolása
    int frekvencia = 22050;
    Uint16 formatum = AUDIO_S16SYS;
    int csatornak = 2;
    int buffer = 4096;

   // az eseménykezeléshez használjuk fel
    int vege = 0;

   // itt már az audio szolgáltatást is
   // inicializálni kell
    if ( SDL_Init(SDL_INIT_AUDIO|SDL_INIT_VIDEO)
➥< 0 )
    {
        std::cout << "Nem tudom indítani az
➥SDL-t: " << SDL_GetError();
```

1. Lista folytatás

```
        exit(1);
    }

    atexit(SDL_Quit);

    // az első lépés, hogy megnyitjuk az adott
    // paraméterekkel a hangeszközt
    if (Mix_OpenAudio(frekvencia,formatum,
➥csatornak,buffer) < 0)
    {
        std::cout << "Nem tudom megnyitni a
➥hangeszközt!\n";
        exit(1);
    }
   // lekérjük, hogy milyen beállításokat tudott
   // eszközölni a hangrendszer
    Mix_QuerySpec(&frekvencia,&formatum,
➥&csatornak);

    // betöltjük a lejátszani kívánt anyagot
    HANGANYAG = Mix_LoadMUS("music.wav");
    Mix_PlayMusic(HANGANYAG,-1);

   // ez mar ismerős kell, hogy legyen
    kepernyo = SDL_SetVideoMode(320,240,0,0);
    while (!vege)
    {
        while(SDL_PollEvent(&esemeny))
        {
            if (esemeny.type == SDL_KEYUP)
            {
                // ESC-re vége a programnak
                if (esemeny.key.keysym.sym ==
➥SDLK_ESCAPE)
                {
                        vege = 1;
                // megállítjuk a lejátszást és
                // felszabadítjuk a hanganyag helyét
                        Mix_HaltMusic();
                        Mix_FreeMusic
➥(HANGANYAG);
                        HANGANYAG = NULL;
                }
            }
            // hogy azért mégse együnk annyi CPU
            // időt
            SDL_Delay(50);
        }
        return 0;
}
```

mondanak el. Az ***SDL_mixer*** könyvtár segítségével ***WAV, MOD, MID, OGG és MP3*** formátumok lejátszására vagyunk képesek. A ***Mix_OpenAudio*** használatával nyitjuk meg a hangeszközünket a paraméterekben megadott formátum lejátszására. A formátum értéke mindig legyen ***AUDIO_S16SYS***. Régebbi ***SDL*** verziókban több variáció is volt, de az elkövetkezendőkben a fejlesztők biztosra mentek és bevezették a ***Mix_QuerySpec*** eljárást, mely visszatér azokkal az értékekkel, melyek ténylegesen használhatóak a hangkártyán. A ***Mix_LoadMUS*** tölti be adott memóriahelyre a lejátszandó anyagot. A lényeges elem a ***Mix_PlayMusic*** melynek első paramétere a hanganyagot tartalmazó memóriaterület mutatója, a második pedig a lejátszások számát adja meg. Ha ennek az értéke -1 akkor ismételgeti az anyagot a rendszer. Az eseménykezelésről már volt szó, azonban van még két új elemünk: ***Mix_HaltMusic*** és ***Mix_FreeMusic***. Az első eljárás leállítja a lejátszást, a második pedig felszabadítja a hangadatok tárolására használt memóriahelyet. Az utóbbinak egy paramétere van mégpedig a felszabadítandó terület mutatója.

A ***Mix_PlayMusic*** párja a ***Mix_PlayChannel*** eljárás, mely annyival több társánál, hogy az első paraméterben a lejátszó csatorna számát adhatjuk meg. Ha -1 értéket adunk meg itt, akkor a rendszer adja automatikusan a csatornát. Ezen eljárás kiválóan alkalmazható két vagy több hanganyag egy időben történő lejátszásához.

Bolygassuk fel az optikai meghajtónkat!

Az ***SDL*** optikai meghajtókat kezelő szolgáltatásai a legszükségesebb elemeket tartalmazzák. Szépen sorban végig fogunk menni mindegyik lehetőségen. Elsősorban valami képet kell kapnunk a rendszerünkhöz csatlakoztatott meghajtókról. Ebben lesz segítségünkre az ***SDL_CDNumDrives*** és az ***SDL_CDName*** függvény. Az elnevezés megint csak igen beszédes, az első lekérdezi az optikai meghajtók számát, a második a rendszerbeli nevüket adja vissza. Ezek alapján már könnyen tudunk olyan programot írni, mely kilistázza rendszerünk optikai meghajtóit, azok nevével együtt (*2. lista*). Mint már megszokhattuk, ebben az esetben is meg kell „nyitni” az eszközt, mielőtt műveleteket hajtanánk végre rajta. Az ***SDL_CDOpen*** függvény segítségével választhatunk ki egy meghajtót, valamit az ***SDL_CDClose*** függvénnyel zárhatjuk le azt. Például nyissuk ki a tálcát a nulladik sorszámú optikai meghajtón (*3. lista*).

Lehetőségünk van kideríteni az ***SDL*** segítségével, hogy adott meghajtóban, milyen típusú adathordozó van, vagy hogy egyáltalán van-e benne valami. Az `SDL_CDStatus` függvény lesz ebben segítségünkre. Paraméterként természetesen a vizsgálandó meghajtó sorszámát várja. A függvény egy `CDStatus` nevű adatstruktúrát ad vissza, mely alapján már tájékozódhatunk a meghajtó állapotáról.

A struktúrát a következő értékek alapján lehet vizsgálni: `CD_TRAYEMPTY`, `CD_STOPPED`, `CD_PLAYING`, `CD_PAUSED`, `CD_ERROR`. Nemsokára látunk egy példát az alkalmazására. Előtte még

2. Lista Optikai meghajtókat kilistázó program

```
#include <iostream>
#include "SDL.h"

int main()
{

    if ( SDL_Init
➥(SDL_INIT_CDROM) < 0 ) {
        std::cout << "Nem tudom
➥indítani az SDL-t:
➥" << SDL_GetError();
        exit(1);
    }

    atexit(SDL_Quit);

    std::cout << "Meghajtók
➥szama: " << SDL_CDNumDrives()
➥<< std::endl;

    for ( int i=0; i<SDL_CDNum
➥Drives(); ++i )
    {
        std::cout << "Meghajtó:
➥" << i << " " <<
➥SDL_CDName(i) << std::endl;
    }

    return 0;
}
```

3. Lista Tálca kinyitása optikai meghajtón

```
/* SDL inicializáció */
...
SDL_CD *cdrom;

cdrom = SDL_CDOpen(0);

SDL_CDEject(0);

SDL_CDClose(cdrom);
...
```

4. Lista Példa audio lemezek kezelésére

```
cdrom = SDL_CDOpen(0);

        // van "valaki" a
        // meghajtóban ?
        if (CD_INDRIVE(SDL_CD
➥Status(cdrom)))
        {
            // cdrom->numtracks
// megadja a lemezen levő
// trackek szamat
            std::cout <<
➥"Trackek száma: " << cdrom-
➥>numtracks << "\n";

            std::cout << "Az
➥első track lejátszása...\n";
            // az első tracket
// lejátszuk
            SDL_CDPlayTracks
➥(cdrom, 0, 0, 1, 0);
        }

    SDL_CDClose(cdrom);
```

tekintsük át, hogy milyen lehetőségeket nyújt az *SDL* az audio anyagot tartalmazó adathordozók kezelésére.

CD-AUDIO támogatása

Az *SDL* négy, kifejezettem audio lemezek kezelését lehetővé tevő függvénnyel rendelkezik. Az első az `SDL_CDPlay`. Ez az egész *CD* lejátszását teszi lehetővé, viszont e mellet létezik még egy nagyon hasonló függvény, az `SDL_CDPlayTracks`, melynek funkciói gazdagabbak és természetesen ugyanazt a funkciót is ellátja mint az előbb említett `SDL_CDPlay`. Ezek miatt inkább erről ejtsünk néhány szót. Az `SDL_CDPlayTracks` rendre a következőket várja paraméterként: a meghajtó száma, kezdő track száma, kezdő frame száma, utoljára lejátszandó track száma, majd ennek a frame száma. A framek audio adategységeket jelölnek. A meghajtó státuszának beolvasása során az *SDL* a `CD_FPS` változóban tárolja számunkra az adott audio lemezhez tartozó egy másodpercre eső framek számát.
Ezek alapján már könnyen megy a következő: játsszuk le a behelyezett audio lemez második sávjának első tíz másodpercét:

```
SDL_CDPlayTracks(cdrom,
➥1,0,0,CD_FPS*10)
```

Az `SDL_CDPause` és az `SDL_CDResume` függvények segítségével, rendre megállíthatjuk illetve folytathatjuk a lejátszást. Mindkét függvény egy `SDL_CD` típusú változót vár paraméterként. Lássunk példaként egy olyan kódot, mely az eddig említett funkciók mindegyikére mutat alkalmazást (*4. lista*).

SDL Timer – Az időzítő

Az optikai meghajtók bűvölése után térjünk át, egy másik igen hasznos témára, az ***időzítők*** használatára.
Az időzítő szolgáltatás tagjai nincsenek sokan: `SDL_GetTicks`, `SDL_Delay`, `SDL_AddTimer`, `SDL_RemoveTimer`. Ezeken kívül még alkalmazható az ***SDL_SetTimer*** funkció is, de ezt a fejlesztők már nem javasolják, mert az elkövetkezendő verziókban ez a funkció már nem fog helyet kapni. Helyette inkább használjuk az említett `SDL_AddTimert`. Lássuk sorjában az időzítő szolgáltatás tagjai.
Az `SDL_GetTicks` függvény az *SDL* inicializálása óta eltelt időt adja vissza ***milliszekundumban***. A visszaadott érték típusa ***Uin32***. Ebből is látszik, hogy nem számolhatja végtelenségig ezt az időt. Körülbelül 49,7 nap után fordul át a *„mutató”*.
Várakozni a program futása közben az `SDL_Delay` utasítással lehetséges, mely egy ***Uint32*** típusú értéket vár paraméterként, melynek értéke a várakozás időtartama megint csak ***milliszekundumban***.
Az egyik legfontosabb az ***SDL_AddTimer***. Segítségével megadott idő intervallumonként hívhatunk meg egy alprogramot. Ennek „párja” az `SDL_RemoveTimer`, mely az előbbi függvény által létrehozott időzítőt szünteti meg.
Ezek alapján, lássunk egy programot mely két másodpercenként kiír egy üzenetet a képernyőre, majd megszünteti az időzítőt és vár még pár másodpercig, hogy tényleg lássuk

5. Lista Az időzített eljárásunk ki fog írni egy üzenetet

```
Uint32 idozito(Uint32
➥intervallum,void *param)
{

    std::cout << "Időzítő
➥üzenete...\n";

    return (intervallum);
}
```

6. lista Példa időzítő alkalmazására

```
...
// inicializáltuk az SDL-t
...
// létrehozzuk az időzítőt, 2
// másodpercenként kapunk
// üzenetet
    SDL_TimerID mytimer =
➥SDL_AddTimer(2000,
➥idozito, NULL);
    // varunk 10 másodpercet,
// addig kapunk üzeneteket
    SDL_Delay(10000);
// töröljük az időzítőt
    SDL_RemoveTimer(mytimer);

    std::cout << "Már nincs
➥timer.\n";
// várunk még egy kicsit, hogy
// lássuk tényleg törölve lett
    SDL_Delay(5000);
...
```

megszűntek az üzenetek. Ez az az eljárás, amit az időzítőnk hívogatni fog (*5. lista*)
Egy időzítő által meghívott eljárásnak teljesítenie kell bizonyos követelményeket. Visszatérési értéke ***Uint32*** típusú kell, hogy legyen. Az első paraméter szinten ilyen típus, és ezt a paramétert vissza kell adnia. Ezek a megkötések az időzítőrendszer implementációjából fakadnak. Most pedig rakjuk össze a programunkat (*6. lista*). Az `SDL_AddTimer` függvény tehát első paraméterként az intervallumot várja (milliszekundum), másodikként a meghívandó eljárás nevét, majd harmadszorra az eljárás által várt paraméterek következnek. Ez utóbbi a mi esetünkben `NULL`, mert nem dolgoz fel paramétereket az időzítőeljárásunk.

SDL végszó

A cikksorozat három részében megismerkedhettünk az ***SDL*** multimédiás programozói könyvtár alapvető elemeivel. Remélhetőleg elég segítséget nyújtottam, ahhoz, hogy az érdeklődők elkezdhessék az ***SDL*** segítségével való fejlesztéseiket. Megtanultuk az eseménykezelés, videó szolgáltatások, optikai meghajtók, időzítők és végül a hangszolgáltatás kezelésének alapjait. Játékfejlesztéshez Linux alatt kiválóan alkalmazható, főleg, hogy együtt használhatjuk az ***OpenGL*** könyvtárral is. Aki ráérzett az ízére, annak sok sikert kívánok a továbbiakban a programozáshoz, hogy sok hasznos multimédiás alkalmazás szülessen a linuxos világot gyarapítva.

Radics Péter