

SDL – Multimédiás programozói könyvtár (1. rész)

Nem tudom ki hogyan van vele, de számomra a linuxos átállás után az első hiányérzetem akkor támadt, mikor valamilyen grafikus alkalmazás fejlesztésébe szerettem volna kezdeni. Sokáig nézelődtem az eszközök között, míg végül kikötöttem az SDL mellett. Ez a programozói könyvtár szinte gyerekjátékká teszi a videómódok, hangszolgáltatás, időzítés programozását Linux alatt. Könnyű használni, népszerű, multiplatform és napjainkban már minden disztribúcióban megtalálható.

Mi is az SDL?

Az SDL a *Simple Directmedia Layer* kifejezés rövidítése. Egy multiplatform programozói könyvtár, elsősorban az egér, billentyűzet, joystick és hangszolgáltatások valamint a video framebuffer kezelésére szolgál. Nagyon jó „barátja” az *OpenGL*-nek is. A rövidítést keverni szokták egy igen komoly kifejezéssel: *Specification and Description Language*. Ez utóbbi egy formális nyelv elnevezése, melyet telekommunikációs rendszerek tervezésére fejlesztettek ki. Maradjunk mi most a jó öreg *Linuxnál* és a multimédiás szolgáltatásoknál, melyek arra várnak, hogy az *SDL* segítségével kiaknázzuk őket.

Támogatottság

Felmerülhet a kérdés, hogy milyen nyelvhez is készült ez a könyvtár. Nagyon jó érzés azt elmondani, hogy szinte minden „divatos” programozási nyelvhez létezik *SDL*. A hivatalos honlapról a következő nyelvekhez tölthetjük le: *Ada, C++, C#, Eiffel, Erlang, Euphoria, Guile, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby*. Véleményem szerinte ez igen barátságos lista. Ha már a támogatott nyelveknél tartunk, akkor érdemes megemlíteni a platformokat is, melyeken az *SDL* futtatható: *Linux, BeOS, MacOS, FreeBSD, Qnx, Amiga, Atari, Symbian OS, Win32*. Mindet nem szeretném

felsorolni, hiszen már ebből is látszik, hogy mennyire széles körben támogatott. Minden szükséges eszköz egy helyen elérhető a projekt honlapján: <http://www.libsdl.org>. A sok nyelv közül a *C++* segítségével szeretnék majd bemutatni pár apróságot, hogy mire lehetünk képesek az *SDL* használatával.

Az SDL család

Az *SDL* több különböző *alrendszerrel* rendelkezik. Ezek az *alrendszerek* végül is önmagunkban is programozói könyvtárak: videó, eseménykezelő, audio, cd-rom audio, szálak és időzítő. Lássuk egyenként, hogy mire is képesek ezek a tagok.

Általános szolgáltatások

Az *SDL* könyvtár tartalmaz egy általános, az *SDL* rendszert kezelő szolgáltatás csoportot. Ezekkel lehet inicializálni és lezárni a rendszert, hibakezelést végezni, valamint információkat lekérdezni az *SDL* alrendszereiről.

Videó

A videó szolgáltatás a legbővebb *alrendszer* az *SDL*-ben. Segítségével bármely videó módot – ha azt a videó hardver támogatja – beállíthatunk 8 bites színmélységtől kezdve. Manipulálhatjuk az adott videó felület gamma, paletta és alpha beállításait és még sok mindent. Az *OpenGL* támogatást is rajta keresztül lehet programozni.

Nehéz is lenne itt pár mondatban összefoglalni, hogy milyen sokféle és színes szolgáltatásokat nyújt. Igyekszem majd példákon keresztül megmutatni az alapokat. Kétség kívül ez a legtöbbet használt *alrendszer* az *SDL*-ben.

Ablakkezelés

Ezekkel a szolgáltatásokkal tudjuk manipulálni az adott videó felület bizonyos tulajdonságait, mint teljes képernyő vagy ablakos mód, ablakfelirat és az ablak ikonja.

Eseménykezelés

Az *SDL* másik lényeges része, melynek segítségével biztosíthatjuk alkalmazásunkban a felhasználói interakciókat. A nagy része *egér- és billentyűzetkezelés*, de tartalmaz *joystickra* vonatkozó elemeket is, bár ennek kezelésére az *SDL* külön szolgáltatás csoporttal rendelkezik.

Audio

Segítségével információkat kaphatunk az adott audio hardverről, *wav* fájlokat tölthetünk a memóriába és játszhatunk le, valamint *konverziókat* eszközölhetünk különböző audio formátumok között (ne értsük félre, csak a *wav* formátumokon belül).

Szálak

A szálak programozása igen hasznos dolog játékfejlesztések esetén. Itt található a *szemaforok*,

szálazonosítókat kezelő szolgáltatásokat. Szálakat hozhatunk létre illetve törölhetünk, közöttük randevúkat intézhetünk. Nagyon nem fogunk belemenni a cikk során, mert a szálak programozása párhuzamos programozási ismereteket követel meg. Erről pedig hatalmas mennyiséget lehetne kifejteni.

Időzítő

A szálak mellett a másik fontos dolog az **időzítő**. Játékfejlesztések során megint csak alap dolognak számít. A cikksorozat folyamán lesznek példák az alkalmazására. Nagy vonalakban láthattuk, hogy milyen szolgáltatásokkal kecsegtet az **SDL**. A cikkben sajnos mindenre nem tudok majd kitérni, de remélhetőleg utat nyitok a megismerés felé, azoknak akik még esetleg nem is hallottak az **SDL**-ről, vagy – programozói körökben divatos szakkifejezéssel élve – még csak „szagolgták” azt.

Inicializálás, előkészületek

Mivel a legtöbb disztribúció alpból tartalmazza az **SDL** könyvtárat, így nagyon nem is térünk ki annak telepítésére. Lássuk, hogy hogyan is néz ki egy egyszerű **SDL**-t felhasználó C++ program (1. lista).

Minden **SDL** alkalmazásnak be kell építenie az **SDL.h**-t. Ez a fejlécállomány fogja tartalmazni az alrendszerre történő hivatkozásokat. Csak azokat az alrendszereket használhatjuk melyeket az `SDL_Init` eljárással inicializáltunk. Ez az eljárás `int` típusú paraméterként és egészet ad vissza. Hiba esetén `-1` értékkel tér vissza, ha minden rendben akkor ez az érték `0`. Hiba esetén az **SDL** hibaüzenetét az `SDL_GetError` függvény adja vissza. Az inicializálás során adhatjuk meg, hogy mely alrendszereket szeretnénk használni. Ezt különböző flagekkel tehetjük meg. Az elnevezések magukért beszélnek, a felhasználható flagek a következők:

`SDL_INIT_TIMER`, `SDL_INIT_AUDIO`, `SDL_INIT_VIDEO`, `SDL_INIT_CDROM`, `SDL_INIT_JOYSTICK`. Ha mindent egyszerre szeretnénk használni, nem fontos beírni az összeset, használjuk inkább az `SDL_INIT EVERYTHING` jelzőt. Megjegyzem ez nagyon kényelmes dolog, de ha az alkalmazásunk optimális teljesítményére szeretnénk

1. lista Az SDL inicializálása

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

int main()
{
    if (SDL_Init
        ↪ (SDL_INIT_VIDEO) < 0)
    {
        std::cerr << "Nem
        ↪ sikerült az SDL
        ↪ inicializálása!
        ↪ Hiba: " <<
        ↪ SDL_GetError();
        exit(1);
    }

    /* Ide jöhet az SDL-t
    ↪ felhasználható kódunk */

    SDL_Quit();

    return 0;
}
```

törekedni, csak azokat az alrendszereket inicializáljuk, melyeket feltétlenül használni fogunk a fejlesztés során.

Például:

```
SDL_Init(SDL_INIT_VIDEO |
SDL_INIT_AUDIO).
```

Eseménykezelésért felelős alrendszert soha nem kell külön elindítanunk, ha a videót inicializáltuk. A videóval együtt az eseménykezelés is automatikusan elindul. Az **SDL** rendszert a program végén az `SDL_Quit` utasítás kapcsolja ki, minden inicializált alrendszert deaktiválva.

Ha most ezt a kódot „bedobjuk” a fordítónak akkor, bizony nem lesz sikerélményünk. Az **SDL** fejléceket nem fogja látni a fordító alpból. Az **SDL** csomaghoz tartozik egy hasznos kis program, mely segít konfigurálni a fordítót. Ennek neve: `sd1-config`. Megfelelően paraméterezve megkapjuk azt a kimenetet, melyet a fordítónk hiányol:

```
sd1-config --libs --cflags
```

A program kimenetét – a héjnak hála – játszva át tudjuk adni a fordítónak. Tehát egy **SDL** program (`main.cpp`) fordítása a következő-

2. lista A videó alrendszer inicializálása és az eseménykezelés alapja

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

int main()
{
    SDL_Surface* sdl_surface;
    SDL_Event sdl_event;
    bool main_loop_exit =
    ↪ false;

    /* Ide jön az
    ↪ inicializálás! */

    sdl_surface =
    ↪ SDL_SetVideoMode(640,
    ↪ 480, 16, SDL_SWSURFACE);

    if (sdl_surface == NULL)
    {
        std::cerr <<
        ↪ "SDL_Surface hiba: "
        ↪ << SDL_GetError();
        exit(1);
    }

    while (!main_loop_exit)
    {
        while (SDL_PollEvent
            ↪ (&sdl_event))
        {
            switch
            ↪ (sdl_event.type)
            {
                case
                ↪ SDL_KEYDOWN:
                    main_loop_exit =
                    ↪ true;
                    break;

                default:
                    break;
            }
        }

        SDL_Quit();

        return 0;
    }
```

képpen történhet, ha mindent szabványosan szeretnénk:

```
g++ `sd1-config --libs --cflags`
-wall -pedantic -ansi main.cpp
```

3. lista Egy pixel megjelenítése (16 bites színmélység esetén)

```
void pixel(SDL_Surface*
↳ surface, int x, int y,
↳ Uint16 color)
{
    Uint16 *p = (Uint16
↳ *)surface->pixels + y *
↳ surface->pitch/2 + x;
    *p = color;
}
```

Most, hogy mindent előkészítettünk készítsünk végre egy olyan programot, ami valamilyen szemmel látható módon alkalmazza az *SDL* szolgáltatásait. Kezdjük a videó alrendszer használatával és az eseménykezeléssel, mivel ezek a domináns csoportok. Egy általános grafikus *SDL* alkalmazás a kezdetek kezdetén a 2. listában látható módon nézhet ki.

Amint láthatjuk már egy alap *SDL* program is több sorra rúghat. Nem zárom ki, hogy kevesebb sorból is ki lehet jönni. Ámde lássuk sorban mit is csinál a 2. listában szereplő kód. Láthatjuk, hogy deklaráltunk egy *SDL_Surface* típusú mutatót. Ez a mutató fog mutatni egy videó felületre, amit az *SDL_SetVideoMode* eljárással alakíthatunk ki. Ez az eljárás rendre a következő paramétereket várja: képernyő szélessége, magassága, bitek száma pixelenként és flagek. Ha gond van akkor *NULL* mutatóval tér vissza. A paraméterek közül a flagek szorulnak némi magyarázatra. Ezek közül is több létezik. Az *SDL_SWSURFACE* alapján a rendszer a videó felületet a rendszer memória területén foglalja le, *SDL_HWSURFACE* esetén pedig a videómemóriában. Ha teljes képernyőn szeretnénk látni az adott felbontást, akkor használjuk az *SDL_FULLSCREEN* flaget. Ezeket a flageket természetesen a vagy („|”) művelettel össze tudjuk kapcsolni, például:

```
SDL_SetVideoMode(640, 480, 16, SDL
↳ _SWSURFACE | SDL_FULLSCREEN)
```

A másik fontos dolog az *eseménykezelés*. A videószoftvar szolgáltatás szemléltetése

kényelmesebb így, hiszen ha a kódot enélkül futtatnánk, akkor a videófelületből nem láthatnánk semmit sem, csupán egy felvillanó ablakot, vagy teljes képernyős mód esetén egy villanást.

Az *SDL* minden eseményt egy eseménysorban tárol. Az ebben a sorban tárolt eseményeket az *SDL_PollEvent* eljárással nyerhetjük ki. Az *SDL_Event* típus egy *unió* típus. Innen az esemény típusától függően kinyerhetjük a számunkra értékes adatokat, többek között, hogy milyen az adott esemény (billentyűleütés vagy felengedés, egérgomb, egér mozgása stb.). Példánkban (2. lista) az *SDL_KEYDOWN* eseményre hivatkozunk. Használhatjuk még az *SDL_MOUSEMOTION*, *SDL_MOUSEBUTTONDOWN*, *SDL_MOUSEBUTTONUP*, szintén eléggé magukért beszélő elnevezéseket is. Tehát alaplól az eseménykezeléshez deklarálnunk kell egy *SDL_Event* változót, valamit ki kell nyernünk ebbe a változóba az eseménysorból, az éppen soron lévő eseményt az *SDL_PollEvent* eljárással. Innen pedig a típustól függően lekezelhetjük a felhasználó interakcióit. Később még több példát is láthatunk erre. Most főként a kényelem miatt vezettük

csak be az eseménykezelést. A programunk most bármely billentyű megnyomására kilép.

Egy színes példa

Térjünk vissza a videószoftvarhoz. Most hogy már van egy felületünk a soron következő dolog, hogy meg is tudjunk jeleníteni valamit. Lássunk hogyan nézhet ki egy pixel megjelenítő rutin (3. lista).

A *p* mutatóban az *x* és *y* koordinátákból kiszámolt pixel memóriában elfoglalt helyét kapjuk meg. Ez most a 16 bites színmélység speciális esete. Más bit értékek esetén ügyeljünk a mutatók típusára! A *surface->pixels* tag mutat a tényleges videófelületre, ahová a színértékeket írhatjuk be. Ehhez adjuk hozzá a klasszikus módon kiszámított pixel helyét. A szín (*color*) érték kiszámításához az *SDL_MapRGB* függvényt hívjuk segítségül. Ezzel az eljárással keverhetünk ki a piros, zöld, kék értékekből, az adott felület típusától függően színértékeket. Például állítsuk elő az *sd1_surface* felülethez illő sárga színértéket:

```
Uint16 sarga = SDL_MapRGB
↳ (sd1_surface->format,
↳ 0xff, 0xff, 0x00)
```



■ 1. ábra Egy egyszerű színátmenet *SDL* segítségével

Így a (10,10) koordinátában már ki tudunk rajzolni egy sárga képpontot:

```
pixel(sd1_surface, 10, 10, sarga)
```

Azonban egy videófelületre nem lehet csak úgy „firkálni”. A felületet le kell zárunk, addig amíg a módosítás történik. Ez több szálú programozás esetén hasznos dolog, hiszen gondoljuk csak el mi történne ha egyszerre több programszál egy időben kezdene kirajzolni egy adott felületre. Az *SDL_LockSurface* eljárás segít nekünk. Lássunk egy konkrét példát, mely a már ismertett *pixel* eljárást alkalmazza (4. lista) és a következő képet fogja kirajzolni: (1. ábra).

Amint látjuk alkalmaztunk egy *SDL_UpdateRect* nevű eljárást. Ennek segítségével aktivizáljuk az adott felületen a változtatásokat és jelenítjük meg a frissített felületet. Paraméterként a felület nevét, majd a frissítendő négyzet koordinátáit várja (bal felső sarok, jobb alsó sarok). Hogy látványosan elnevezzük az első komolyabb *SDL* programunkat, ismerjük meg az ablakkezelő rendszer egyik funkcióját:

```
SDL_WM_SetCaption.
```

Ezen eljárás segítségével képesek vagyunk módosítani az adott alkalmazás ablakának feliratát, valamint ikonnevét. Például:

```
SDL_WM_SetCaption("SDL  
↳ Színátmenet", "")
```

Ezt a sort az *sd1_surface* inicializálása után érdemes beszúrni a fenti példában.

Remélhetőleg érdekes információkkal szolgált az olvasnivaló az *SDL* programozói könyvtárról. Még korántsem láttunk mindent! A következő részben még tárgyalunk néhány videó funkciót és eseménykezelést. Rajzolunk majd az egér segítségével és példát láthatunk majd egy *BMP* fájl megjelenítésére is. Később használni fogjuk az audio, cd-rom, timer alrendszer funkcióit is.

Addig is érdemes ellátogatni a <http://www.libsdl.org> oldalra

4. Lista Színátmenet megjelenítése

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

void pixel(SDL_Surface* surface, int x, int y, Uint16
↳ color){...}

int main()
{
    SDL_Surface* sd1_surface;
    SDL_Event sd1_event;
    bool main_loop_exit = false;
    Uint16 color;

    ...
    /* Itt inicializálunk */

    sd1_surface = SDL_SetVideoMode(255, 255, 16,
↳ SDL_SWSURFACE);

    if (sd1_surface == NULL) { ... }

    // Lefoglaljuk az írásjogot a felületre.
    SDL_LockSurface(sd1_surface);

    for(Uint16 i=0; i<255; i++)
    {
        for(Uint16 j=0; j<255; j++)
        {
            // Színkeverés és kirajzolás.
            color = SDL_MapRGB(sd1_surface->format, i,
↳ j, 255-j);
            pixel(sd1_surface, i, j, color);
        }
    }
    // A felület frissítése
    SDL_UpdateRect(sd1_surface, 0, 0, 255, 255);

    // Elengedjük a felületet.
    SDL_UnlockSurface(sd1_surface);

    while (!main_loop_exit) {...}

    ...

    return 0;
}
```

bővebb információkért és „lapozgatni” a <http://www.libsdl.org/cgi/docwiki.cgi> oldalakat. Itt többek között utána lehet nézni az inicializációs flageknek valamint a videómódok beállításánál használt flageknek is. Az események típusai is részletesen megtalálhatóak, melyekből természetesen a következő számban egy párral meg is fogunk ismerkedni.



Radics Péter

(peter.radics@gmail.com)
Az ELTE-n tanuló programtervező matematikus szakon. Hobbim a kosárlabda, autóvezetés, web-design, programozás. Főleg webes alkalmazások fejlesztése érdekel. 4 éve megrögzött Linux felhasználó vagyok.