Tutorial Switch sobre PORTLIBS

Curso 2k22/2k23

Apellidos, nombre	Arnau Calatayud, Jordi (jarncal@inf.upv.es)
Titulación	Grado de Ingeniería informática
Fecha	04/05/2023

UNIVERSIDAD POLITECNICA DE VALENCIA

Índice

UNIVERSIDAD POLITECNICA DE VALENCIA

Índice de figuras

UNIVERSIDAD POLITECNICA DE VALENCIA

1 Resumen de las ideas clave

En este documento vamos a presentar algunas de las librerías llamadas *portlibs*. Las *portlibs* son aquellas librerías que han sido desarrolladas en el entorno de escritorio y han sido portadas a las distintas plataformas para que sean utilizadas en cualquiera de ellas, pero en este documento vamos a ver cómo usarlas en la *Switch*.

Como estas librerías han sido desarrolladas fuera de los entornos tan cerrados de las videoconsolas, las funciones de estas librerías van a ser comunes en todas las plataformas en las que estén disponibles, cambiando, eso sí, las funciones que son específicas de cada consola (el include de switch.h en el caso de Switch, por ejemplo).

Las *portlibs* son librerías muy diversas: las funciones que realizan cada una son muy diferentes unas a otras y aunque haya algunas cuyo propósito sea el mismo la manera en la que llegan a una solución es posiblemente muy diferente a las demás. Existen librerías que nos facilitan un acceso al hardware para cualquier plataforma (SDL), otras decodificadoras de audio (FLAC...), otras que simulan las físicas de objetos 2D (box2d), etc.

2 Introducción

Las librerías que se han decido abordar en este documento son las siguientes: box2d, SDL, opus y mikmod. Son tan pocas las librerías que se van a estudiar por qué en la instalación de devkitpro no se instalaron las *portlibs* y automáticamente y se estuvieron varias semanas que deberían haber estado invertidas en avanzar con el estudio que se perdieron. No obstante, la librería de SDL es una muy amplia y toca vértices muy dispares (audio, impresión por pantalla, controles…) ya que maneja el hardware, por lo que este documento va a contener diversas funcionalidades.

3 Objetivos

A partir del estudio de los ejemplos que se abordan en este documento, el lector será capaz de:

- Compilar cualquier *portlib* para *Switch*, independientemente de si está en el pkgconfig o no.
- Entender de forma superficial el funcionamiento de algunas de estas librerías.

El presente documento está encaminado a ofrecer una perspectiva inicial de cómo usar alguna de estas librerías. No es el objetivo realizar un estudio e implementación exhaustiva de las librerías presentadas con el fin de entender la librería al completo, sino una pincelada de qué cosas se pueden hacer en estas librerías y quizás qué cosas se podrían realizar con el uso de las mismas.

4 Desarrollo

Para llevar a cabo los ejemplos descritos en este documento necesitaremos instalar devkitpro y las *portlibs*, que en principio deberían de instalarse junto con el kit de desarrollo. Si no es así se deberá usar el instalador de paquetes *pacman* que ofrece devkitpro. Dado que el objetivo de este documento no es que se sepa instalar el SDK y las librerías portadas, no se profundizará en este tema.

4.1 Compilación de los ejemplos

Una vez instalado lo anteriormente citado, necesitaremos poder usar las *portlibs*. Para ello, tomaremos como base cualquier fichero Makefile disponible en los ejemplos devkitpro para la switch. Prácticamente todo el Makefile se queda como está, excepto la parte en la que se incluyen las librerías que se quieren utilizar, obviamente.

Primeramente, comprobamos si la librería que queremos usar tiene asociado un pkg-config, que nos facilita la inclusión de la librería en el ejecutable final porque contiene información adicional de las librerías que se van a utilizar e incluye aquellas librerías adicionales que se puedan usar dentro de la librería que se quiere usar (por ejemplo, si queremos importar la librería box2d, necesitamos incluir también la librería libLinearMath, cosa que ya hace el pkg-config y nos facilita la instalación). Para ver si la librería a utilizar tiene asociado o no un pkg-config tenemos que ver en la carpeta donde están las libs de portlibs una llamada pkgconfig. Si la librería que se quiere usar tiene un archivo con el mismo nombre y extensión ".pc" significa que hay un pkgconfig asociado.

```
PS C:\devkitPro\portlibs\switch\lib\pkgconfig> ls


    Directorio: C:\devkitPro\portlibs\switch\lib\pkgconfig


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        20/04/2023     11:21            314 bullet.pc
-a----        20/04/2023     11:21            230 egl.pc
-a----        20/04/2023     11:21            252 expat.pc
-a----        20/04/2023     11:21            267 flac++.pc
-a----        20/04/2023     11:21            278 flac.pc
-a----        20/04/2023     11:21            419 freetype2.pc
-a----        20/04/2023     11:21            321 fribidi.pc
-a----        20/04/2023     11:21            447 gdlib.pc
-a----        20/04/2023     11:21            229 glapi.pc
-a----        20/04/2023     11:21            242 glesv1_cm.pc
-a----        20/04/2023     11:21            233 glesv2.pc
-a----        20/04/2023     11:21            377 glfw3.pc
-a----        20/04/2023     11:21            267 jansson.pc
-a----        20/04/2023     11:21            272 json-c.pc
-a----        20/04/2023     11:21            330 libarchive.pc
-a----        20/04/2023     11:21            342 libass.pc
```

Figura 1 Lista de algunos pkgconfig disponibles

En el caso de que sí tenga pkgconfig asociado, en el Makefile en el apartado de CFLAGS, habrá que poner lo que aparece en la Figura 2, poniendo tras la etiqueta –cflags el nombre de la librería que se quiere usar en el programa. Además, en el apartado LIBS se pondrá lo descrito en la Figura 3, cambiando también las librerías tras la etiqueta –libs. Todo lo demás quedaría igual y el ejecutable debería de poder generarse sin mayor complicación.

```
CFLAGS  :=  `$(PREFIX)pkg-config --cflags sdl2 SDL2_mixer SDL2_image` -Wall -O2 -ffunction-sections \
            $(ARCH) $(DEFINES)

CFLAGS  +=  $(INCLUDE) -D__SWITCH__
```

Figura 2 Cambios a CFLAGS

```
LIBS    :=  `$(PREFIX)pkg-config --libs sdl2 SDL2_mixer SDL2_image SDL2_ttf` \
            -lnx
```

Figura 3 Cambios a LIBS

Por otra parte, en el caso de que la librería a emplear no tiene asociado ningún pkgconfig que nos facilite la compilación de estas, tendremos que añadirlas manualmente. La manera de hacerlo es viendo en la lista de las librerías el nombre de la que se quiere compilar y, en el apartado LIBS del Makefile añadir -l junto con el nombre de la librería sin el prefijo lib, tal y como se observa en la Figura 4.

```
LIBS    :=  -lnx -lbox2d
```

Figura 4 Cambios a DIRS sin pkgconfig

Es importante saber que si se introducen librerías en el Makefile estas estarán incluidas en el ejecutable final independientemente de si son usadas o no, por lo que es vital introducir solo las librerías necesarias para liberar espacio no utilizado.

4.2 SDL

Pasamos ya a hablar de los ejemplos de implementación de las *portlibs* comentadas anteriormente. SDL (Simple DirectMedia Layer) es una librería diseñada para proveer acceso al hardware via OpenGL y Direct3D. Como hemos comentado anteriormente, podemos acceder al audio, botones, pantalla, etc. El ejemplo que se ha utilizado para expilcar esta librería ha sido un clásico *Pong*.

Como existen varios flancos dentro de la misma librería vamos a ir comentándolos paso por paso.

4.2.1 Pantalla

En este apartado se explicará todo lo relacionado con ver por pantalla tanto texto como imágenes.

Primero de todo, para renderizar tanto texto como imágenes definimos los *SDL_Texture*, que son los objetos que se van a mostrar por pantalla (más tarde veremos cómo) y los SDL_Rect que corresponde con la posición en la pantalla en la que se observará ese objeto.

```
SDL_Texture *bola_tex = NULL, *pala1_tex = NULL, *pala2_tex = NULL, *resultado_tex = NULL;
SDL_Rect pos_bola = { 0, 0, 0, 0 }, pos_pala1 = { 0, 0, 0, 0 }, pos_pala2 = { 0, 0, 0, 0 };
```

Figura 5 Definición SDL_Texture y SDL_Rect

Posteriormente, definimos la velocidad que va a tener la pelota del *Pong* e inicializamos todo lo relacionado con la pantalla: SDL_Init con las constantes para inicializar video y timer, IMG_Init para poder observar imágenes y TTF_Init() para poder cargar alguna fuente de texto.

```
int vel_x = rand_range(2, 3);
int vel_y = rand_range(2, 3);

SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);
IMG_Init(IMG_INIT_PNG);
TTF_Init();
```

Figura 6 Inicialización SDL pantalla

Seguidamente, creamos la ventana con los parámetros que se deseen, pudiendo elegir la resolución de la pantalla y el renderer que crea un contexto de renderizado 2D a la pantalla definida previamente.

```
SDL_Window* window = SDL_CreateWindow("Pong", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, SCREEN_W, SCREEN_H, SDL_WINDOW_SHOWN);
SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
```

Figura 7 Crea ventana y renderer

Tras crear la ventana, definimos las imágenes que se quieren renderizar, en este caso la pelota y las dos palas. Para ello cargamos del directorio en el que estén las imágenes una SDL_Surface, definimos su posición y, usando estas imágenes creamos la *SDL_Texture* que es la que se va a renderizar con la función *SDL_CreateTextureFromSurface*. Finalmente, tras crear la SDL_Texture correspondiente ya no es necesario almacenar en memoria las *SDL_Surface*, por lo que se liberan con la función *SDL_FreeSurface* (Figura 8).

Hacemos lo mismo con el texto (Figura 9), pero en este caso para crear la textura en base al texto usamos la función *render_text* que básicamente carga la fuente que se quiere usar, crea un *SDL_Surface* con esta y crea la textura que se va a observar por pantalla. Finalmente, como ya se ha creado la textura libera espacio eliminando la fuente cargada y la *SDL_Surface* (Figuras 9 y 10).

```
SDL_Surface *bola = IMG_Load("data/box.png");
if (bola) {
    pos_bola.x = SCREEN_W / 2 - bola->w / 2;
    pos_bola.y = SCREEN_H / 2 - bola->h / 2;
    pos_bola.w = bola->w;
    pos_bola.h = bola->h;
    bola_tex = SDL_CreateTextureFromSurface(renderer, bola);
    SDL_FreeSurface(bola);
}

SDL_Surface *pala1 = IMG_Load("data/pala.png");
if (pala1) {
    pos_pala1.w = 35;
    pos_pala1.h = 125;
    pos_pala1.x = 0;
    pos_pala1.y = SCREEN_H / 2- pos_pala1.h / 2;
    pala1_tex = SDL_CreateTextureFromSurface(renderer, pala1);
    SDL_FreeSurface(pala1);
}

SDL_Surface *pala2 = IMG_Load("data/pala.png");
if (pala2) {
    pos_pala2.w = 35;
    pos_pala2.h = 125;
    pos_pala2.x = SCREEN_W - pos_pala2.w;
    pos_pala2.y = SCREEN_H / 2 - pos_pala2.h / 2;
    pala2_tex = SDL_CreateTextureFromSurface(renderer, pala2);
    SDL_FreeSurface(pala2);
}
```

Figura 8 Definición de imágenes

```
sprintf(score, "%d - %d", score1, score2);
const char* result = score;
SDL_Rect helloworld_rect = { SCREEN_W / 2, 36, 0, 0 };
resultado_tex = render_text(renderer, result, colors[1], &helloworld_rect);
```

Figura 9 Definición de texto

```
SDL_Texture * render_text(SDL_Renderer *renderer, const char* text, SDL_Color color, SDL_Rect *rect)
{
    TTF_Font* font = TTF_OpenFont("data/LeroyLetteringLightBeta01.ttf", 36);
    SDL_Surface *surface;
    SDL_Texture *texture;

    surface = TTF_RenderText_Solid(font, text, color);
    texture = SDL_CreateTextureFromSurface(renderer, surface);
    rect->w = surface->w;
    rect->h = surface->h;

    SDL_FreeSurface(surface);
    TTF_CloseFont(font);
    return texture;
}
```

Figura 10 Crear textura en base a texto

4.2.2 Audio

Seguidamente, vamos a comentar de qué manera se puede reproducir audio usando la librería de SDL.

Hay dos maneras de definir una salida de audio dependiendo de si queremos que el sonido sea estático o no. En el caso de que queramos que la música esté de fondo podemos usar el tipo de datos Mix_Music y si queremos que la música vaya cambiando dependiendo de cualquier variable (en el caso del Pong, al chocar con las paredes) usaremos Mix_Chunk, que define un "array" de sonidos.

```
Mix_Music *music = NULL;
Mix_Chunk *sound[4] = { NULL };
```

Figura 11 Definición audio

Tras definir los distintos tipos de audio que se van a reproducir, pasamos a iniciar el subsistema de SDL encargado del audio con la función *SDL_InitSubSystem* y la constante *SDL_INIT_AUDIO*. Después definimos la cantidad de canales de audio que se reproducirán, en este caso 5 y abrimos el audio pasando por parámetros algunas especificaciones del formato de salida de audio como la frecuencia, el tipo, el número de canales (1 – mono, 2 – estéreo) y el tamaño del buffer de audio con la función Mix_OpenAudio. Finalmente, antes de entrar al bucle asociamos a cada variable previamente definida el audio correspondiente y, como la música de fondo sonará lo que dure la ejecución, reproducimos ya la música con la función Mix_PlayMusic pasándole como parámetro la variable que contiene el audio que se quiere reproducir (Todo este párrafo se encuentra en la Figura 12).

```
SDL_InitSubSystem(SDL_INIT_AUDIO);
Mix_AllocateChannels(5);
Mix_OpenAudio(48000, AUDIO_S16, 2, 4096);

music = Mix_LoadMUS("data/background.ogg");
sound[0] = Mix_LoadWAV("data/pop1.wav");
sound[1] = Mix_LoadWAV("data/pop2.wav");
sound[2] = Mix_LoadWAV("data/pop3.wav");
sound[3] = Mix_LoadWAV("data/pop4.wav");
if (music)
    Mix_PlayMusic(music, -1);
```

Figura 12 Inicialización música pre-bucle

En el caso del audio dinámico, simplemente se toma un valor del 0 al 3 (correspondiente a los cuatro canales dinámicos y, mediante la función Mix_PlayChannel se reproduce el audio asociado ese número.

```
snd = rand_range(0, 3);
if (sound[snd])
    Mix_PlayChannel(-1, sound[snd], 0);
```

Figura 13 Cambio audio dinámicamente

Finalmente, al terminar la ejecución, para liberar espacio se usa la función Mix_HaltChannel, Mix_FreeMusic que paran la ejecución de todos los canales (si se pone por parámetro -1) y se libera el espacio del Mix_Music definido, respectivamente. Posteriormente, se recorre el array de sonidos y se libera cada chunk con Mix_FreeChunk y se cierra el SDL de audio con Mix_CloseAudio y Mix_Quit.

```
Mix_HaltChannel(-1);
Mix_FreeMusic(music);
for (snd = 0; snd < 4; snd++)
    if (sound[snd])
        Mix_FreeChunk(sound[snd]);

Mix_CloseAudio();
Mix_Quit();
```

Figura 14 Liberación espacio música

4.2.3 Controles

Para poder usar los controles se puede hacer de dos maneras: definiendo un SDL_Joystick que simule los controles de la Switch (de la manera en la que se ha implementado en el Pong) o definiendo un SDL_Event que espere un evento y a partir de ahí controlar qué tecla se ha pulsado.

Si eligiésemos la segunda opción, la manera de de escuchar eventos es mediante un bucle dentro del bucle appletMainLoop con SDL_PollEvent(&event) como guarda, siendo evento un SDL_Event previamente definido.

Por otra parte, si elegimos la primera opción, primero inicializamos el subsistema encargado del manejo del joystick, lo activamos y definimos un SDL_Joystick con la función SDL_JoystickOpen.

```
SDL_InitSubSystem(SDL_INIT_JOYSTICK);
SDL_JoystickEventState(SDL_ENABLE);
SDL_Joystick* controller = SDL_JoystickOpen(0);
```

Figura 15 Inicialización del joystick

Seguidamente, dentro del bucle principal se comprueban los botones que interesa saber si están presionados o no. Para hacerlo, llamamos a la función SDL_JoystickGetButton pasándole el joystick y el botón que se quiere comprobar. Esto devuelve un Uint8 que está a 1 si el botón está activo y a 0 en caso contrario.

```
if (controller) {
    upButton1 = SDL_JoystickGetButton(controller, JOY_A);
    downButton1 = SDL_JoystickGetButton(controller, JOY_B);
    upButton2 = SDL_JoystickGetButton(controller, JOY_UP);
    downButton2 = SDL_JoystickGetButton(controller, JOY_DOWN);
}
```

Figura 16 Comprobar botones presionados

Finalmente, como en todos los casos, se libera el espacio en memoria. En este caso, como inicializa un subsistema de SDL, tan solo será necesario cerrar el joystick previamente abierto (SDL se para con SDL_Quit).

```
SDL_JoystickClose(controller);
```

Figura 17 Cierre joystick

4.3 Box2D

La segunda *portlib* que vamos a comentar en este documento es box2D. Box2D es una librería que simula las físicas de objetos 2D, como las colisiones y la gravedad.

La base para crear una aplicación con box2d es crear un mundo con una gravedad que gobernará todos los objetos que se definan dentro de este.

```
b2Vec2 gravity(0.0f, 8.0f);

std::unique_ptr<b2World> world = std::make_unique<b2World>(gravity);
```

Figura 18 Creación del mundo

Posteriormente, para el ejemplo que tenemos para observar cómo funcionan las colisiones, tenemos un objeto estático en el suelo y un objeto dinámico que cae sobre este. Para definir los objetos tanto dinámico como estático se sigue la misma estrategia: primero definimos la posición, el tipo, etc. en un b2BodyDef, luego creamos un cuerpo con las definiciones realizadas anteriormente y, finalmente, se crea un elemento con las dimensiones y demás atributos adicionales como la densidad del objeto, fricción, forma, elasticidad…

```
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0.0f, SCREEN_HEIGHT);

b2Body *groundBody = world->CreateBody(&groundBodyDef);

b2PolygonShape groundBox;

groundBox.SetAsBox(320, 32);

groundBody->CreateFixture(&groundBox, 0.0f);
```

Figura 19 Definición de cuerpo estático

```
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.position.Set(152.0f, 0.0f);
b2Body* body = world->CreateBody(&bodyDef);

b2PolygonShape dynamicBox;
dynamicBox.SetAsBox(16.0f, 16.0f);

b2FixtureDef fixtureDef;
fixtureDef.shape = &dynamicBox;

fixtureDef.density = 1.0f;

fixtureDef.friction = 0.8f;

body->CreateFixture(&fixtureDef);
```

Figura 20 Definición de cuerpo dinámico

Posteriormente, en el bucle while, para que cada iteración cada objeto avance un paso se usa la función del mundo llamada Step que mueve cada objeto dentro de este mundo dependiendo de su posición, gravedad, colisiones… Y para poder mostrar por pantalla las condiciones en las que se encuentra cada objeto en cada iteración se hace uso de las funciones de los cuerpos llamadas GetPosition y GetAngle que recogen la posición y el ángulo del objeto, respectivamente.

```
position = body->GetPosition();
angle = body->GetAngle();

pos_cuadrado.x = position.x;
pos_cuadrado.y = position.y;

b2Vec2 groundPos = groundBody->GetPosition();
```

Figura 21 Conseguir posición y ángulo

4.4 Mikmod

Esta librería es un reproductor de audio que soporta varios formatos (mod, s3m, it y xm). No obstante, en el ejemplo de estudio no se reproduce ningún audio. En el ejemplo simplemente se registran todos los drivers y muestra por pantalla todos los drivers que contiene la librería, y hace lo mismo con los formatos de audio que soporta.

Las funciones que se utilizan son MikMod_RegisterAllDrivers para registrar los driver y MikMod_InfoDriver para mostrar información de los drivers registrados. De la misma manera existe MikMod_RegisterAllLoaders y MikMod_InfoLoader que hacen lo mismo pero con los formatos de audio que es capaz de reproducir.

```
char *informacion;

MikMod_RegisterAllDrivers();
informacion = MikMod_InfoDriver();
printf("libMikMod registered drivers: \n%s\n", informacion);
MikMod_free( informacion );

MikMod_Init("");

MikMod_RegisterAllLoaders();
informacion = MikMod_InfoLoader();
printf("libMikMod registered loaders: \n%s\n", informacion);
MikMod_free( informacion );

MikMod_Exit();
```

Figura 22 Funciones mikmod

Figura 23 Salida por pantalla mikmod

5 Conclusión

Como conclusión, podemos decir que las portlibs son unas liberías con funciones muy distintas cuya único parecido es que pertenecen al mundo del computador de escritorio y han sido portadas a las diferentes plataformas para facilitar y unificar algunas funciones específicas.

Como conclusión del trabajo decir que empecé demasiado tarde por no saber instalar las portlibs y no haberse instalado junto con devkitpro sin tener mucha idea del desarrollo de videojuegos y, por descontado, de switch. He visto muchos ejemplos tanto en devkitpro como por la red, pero al portar estos ejemplos fue muy complicado porque no entendía las librerías básicas de switch y se me hizo un mundo. Además, para entender cómo compilar estas librerías no lo entendí hasta muy adelante.

6 Bibliografía

Bibliografía y referencias, desglosadas o no en apartados.

[1] Contenedor varias librerías: https://jwu.github.io/wiki/gamedev-libs/

[2] SDL2: https://www.libsdl.org/

[3] Box2D: https://box2d.org/documentation/md__d_1__git_hub_box2d_docs_hello.htmlç

[4] Mikmod: https://mikmod.sourceforge.net/

[5] Portlibs: https://devkitpro.org/wiki/portlibs

[6] Ejemplos de años anteriores, especialmente el de Monogame vs devKitPro

[7] Bullet (no usada): https://pybullet.org/wordpress/index.php/forum-2/

[8] xml2 (no usada): https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home

[9] Fribidi (no usada): https://github.com/fribidi/fribidi

[10] Ejemplo a portar fribidi (no usado): https://mikmod.sourceforge.net/