

Títol: Entornos libres para el desarrollo de videojuegos: un caso práctico.

Volum: 1

Alumne: Raúl González Moya

Director/Ponent: Antoni Soto i Riera

Departament: LSI

Data:

DADES DEL PROJECTE

Títol del Projecte: Entornos libres para el desarrollo de videojuegos: un caso práctico.

Nom de l'estudiant: Raúl González Moya

Titulació: Ingeniería Técnica Informática de Gestión.

Crèdits: 22,50

Director/Ponent: Antoni Soto i Riera

Departament: LSI

MEMBRES DEL TRIBUNAL (nom i signatura)

President: Josep Vilaplana Pastó

Vocal: Manuel Alejandro Pajuelo González

Secretari:

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Estructura del documento	2
2. Análisis	5
2.1. Hardware de la PS2 y carga de software	5
2.2. Carga de software Oficial	7
2.2.1. Modchips	8
2.2.2. Discos de arranque	10
2.2.3. Carga a través de disco duro mediante discos de arranque	11
2.3. Carga de software NO oficial	11
2.3.1. Disco duro	11
2.3.2. El emulador PCSX2-0.8.1	12
2.3.3. Loaders	13
2.4. Entornos de desarrollo para PS2	14
2.4.1. Kit oficial de desarrollo de Sony	14
2.4.2. Kit de GNU/Linux para PS2	14
2.4.3. Programación RAW	16
2.5. Compilación cruzada	16
2.5.1. Canadian Cross	17
2.6. Autotools	18
2.7. Simple DirectMedia Layer (SDL)	19
3. Entorno de desarrollo para la PS2	21
3.1. Sistema operativo	21

3.2. Software para desarrollar	21
3.3. Entorno de programación RAW (PS2DEV)	22
3.4. Librerías SDL	23
3.5. El método de carga de software NO oficial	24
4. Portando un juego a la PS2	27
4.1. Tests básicos del entorno PS2DEV	27
4.2. Test del port de SDL con un proyecto PS2 en SDL	29
4.3. Test de portabilidad básicos	31
4.4. Cuadrado	31
4.5. Círculo	32
4.6. Teclas	32
4.7. Colisiones	33
4.8. Tests de entrada/salida	33
4.9. hello.c	33
4.10. helloFile.c	34
4.11. helloFile.c	35
4.12. helloFileToFile.c	35
4.13. Test de portabilidad de un proyecto SDL que utiliza autotools	36
4.13.1. Autotools y compilación cruzada	38
5. Valoración económica	49
5.1. Costes de hardware y software	49
5.2. Costes de dedicación	50
5.3. Detalles de las tareas realizadas	51
6. Conclusiones	53

Índice de figuras

2.1. Kit oficial de GNU/Linux para PS2	15
2.2. Diagrama de entorno de compilación cruzada.	17
2.3. Diagrama de entorno canadian cross.	18
4.1. Pongix versión SDL de Pong para Linux	37

Capítulo 1

Introducción

1.1. Motivación

El motivo de porqué escogemos este Proyecto Final de Carrera(PFC) es porque queremos probar si somos capaces de portar un videojuego desde GNU/Linux [LIN2002] a la videoconsola PlayStation 2(PS2) [PS22000]. Además nos resulta interesante la temática de este proyecto ya que se aparta un poco de la temática del resto de proyectos más estándar finales de carrera, cosa importante tener en cuenta, ya que esto es una dificultad añadida a la hora de realizar el PFC.

Por otro lado la decisión de realizar el PFC usando software libre [FSF1996] siempre resultó interesante por las ventajas que ofrece el software libre respecto al propietario:

- **El bajo o nulo** coste de los productos.
- **La libertad de uso y redistribución** tantas veces y en tantos ordenadores como el usuario desee.
- **Independencia tecnológica.** Teniendo acceso al código fuente permite el desarrollo de nuevos productos sin la necesidad de desarrollar todo el proceso partiendo de cero.
- **Los formatos estándar** permiten una interoperatividad más alta entre sistemas, evitando incompatibilidades.

Realizar el PFC utilizando herramientas con licencia de software libre, permite que podamos distribuir todas las aplicaciones que utilizamos. Así otros usuarios que estén interesados en este tema pueden empezar teniendo una base, evitando los problemas que nosotros hemos tenido.

1.2. Objetivos

El objetivo principal de este PFC es el estudio de entornos libres para el desarrollo de videojuegos.

La metodología del proyecto consiste en centrar el estudio de los entornos libres de desarrollo en un caso práctico: la videoconsola Playstation 2 de Sony [SON1945].

Las tareas que llevamos a cabo para realizar el proyecto son las siguientes:

Búsqueda y elección de un entorno de desarrollo

La primera tarea que realizamos fue buscar información sobre todos los entornos de desarrollo disponibles para programar videojuegos en PS2.

De los entornos encontrados descartamos todos los que no son software libre y del resto realizamos la implantación de cada uno de ellos en PC con una distribución de GNU/Linux [GNU1996] [LIN2002] y además también estudiamos las características que nos ofrecen cada uno de ellos para así escoger el entorno que más se adecúe a las necesidades del proyecto.

Estudio del entorno de desarrollo

Una vez tenemos escogido el entorno de desarrollo (para este caso práctico el PS2SDK [PS2SDK]), nos disponemos a instalarlo en un PC, con una distribución Ubuntu 7.04 Feisty Fawn [UBU2004] recién instalada, documentando con detalle el proceso que seguimos.

La siguiente tarea es comprobar que el entorno está instalado y configurado de manera correcta. Para probarlo compilamos el videojuego Super Mario Wars [SMW2006], ya programado en Simple DirectMedia Layer(SDL) [SDL2006], del que disponemos una versión de código fuente compatible para PS2. De este modo descartamos posibles errores de configuración del PS2SDK.

Estudio de portabilidad entre dos plataformas y puesta en marcha

Una vez tenemos el entorno preparado la siguiente tarea es estudiar la portabilidad de videojuegos para PS2 tan solo utilizando de forma exclusiva el port que la PS2 tiene de las librerías SDL.

El siguiente paso que realizamos es estudiar que herramientas tenemos disponibles para poder realizar portabilidad de aplicaciones entre dos plataformas.

Ya conocidas las herramientas que disponemos, la siguiente tarea es hacer todas las pruebas necesarias para llevar a cabo la portabilidad de Pongix [PGX2006] el videojuego que escogemos, y además también documentar todos los resultados que obtenemos.

1.3. Estructura del documento

A lo largo de este documento explicamos toda la información que recolectamos, así como las herramientas encontradas, para qué sirven y cómo las configuramos para poder utilizarlas.

En el capítulo 2 realizamos un análisis de toda la tecnología que envuelve este proyecto, en él encontramos una descripción del hardware que dispone PS2, los diversos métodos de carga de

backups de programas oficiales y no oficiales de PS2 y también los entornos de programación disponibles para PS2. Otros aspectos que también vemos analizados en este capítulo son las herramientas básicas necesarias para desarrollar un videojuego. También hacemos un análisis sobre la portabilidad de aplicaciones en el capítulo 2.

En el capítulo 3 explicamos como implantar en una distribución Ubuntu el entorno escogido entre todos los entornos analizados en los capítulos 2. También en esta sección encontramos la configuración de las librerías básicas y de las específicas para desarrollar la portabilidad (las SDL). Además explicamos todas las pruebas que realizamos para dejar listo el entorno.

El capítulo 4 de la memoria está completamente dedicado a todos los experimentos realizados en el estudio de portar Pongix, las dificultades encontradas, los resultados obtenidos y a las conclusiones que obtenemos sobre estos resultados.

A continuación en el capítulo 5 de la memoria encontramos la valoración económica de los gastos de hardware y software así como el tiempo dedicado al proyecto.

Y finalmente, en el capítulo 6 repasamos los objetivos propuestos así como las tareas realizadas para cubrir esos objetivos. Además también hacemos una explicación de qué tareas cumplimos, cuales no y porqué.

Capítulo 2

Análisis

En este capítulo hacemos un análisis breve de la arquitectura de PS2 así como del hardware de que dispone.

Además explicamos los métodos para cargar cualquier tipo de software en ella y también explicamos los entornos de programación libres que hay disponibles para desarrollar en la PS2.

También encontramos un estudio de la compilación cruzado metodología escogida para llevar a cabo la portabilidad de Pongix.

2.1. Hardware de la PS2 y carga de software

Modelos de PS2 hasta la actualidad hay disponible dos distintos la PS2 y la PSTwo también conocida como PS2 Slim debido a la gran reducción de tamaño respecto a su predecesora. Este proyecto se ha realizado con una PS2.

La estructura interna de la PS2 se compone principalmente de cuatro partes:

Emotion Engine (EE)

Éste es el corazón de la PS2, y la parte que la hace distinta a las demás consolas de su misma generación.

Se encarga de realizar las siguientes funciones:

- *Cálculos geométricos* transformaciones, translaciones, etc.
- *Comportamiento del mundo 3D* de la inteligencia artificial(IA) de los personajes, colisiones y en general la física del mundo que se está simulando.
- *Funciones generales* control del programa y en general la dirección del juego.

El resultado del trabajo del Emotion Engine son display lists, esto es, secuencias de comandos de rendering que son enviados al sintetizador gráfico (GS).

En las versiones V13 y V14 de la PSTwo el EE y el GS los unificaron en un solo microchip.

Sintetizador Gráfico (GS)

Es el encargado de recoger los display list que le envía el EE y los representa en la pantalla. El GS es lo que sería la “tarjeta gráfica” de la PS2, como comentamos antes, en las últimas versiones de PSTwo estan unidos ambos chips.

Procesador de sonido (SP)

Es la “tarjeta de sonido” de la PS2. Es capaz de reproducir sonido digital 3D, por su salida óptica digital.

Procesador de Entrada/Salida (IOP)

Es el encargado de manejar los puertos USB, el puerto FireWire, y todo el tráfico de los mandos de control(DualShock [[DSK2004](#)]) del juego y de las tarjetas de memoria.

En las últimas versiones de PS2 y en todas las versiones de PSTwo el puerto del FireWire fue suprimido, e incorporaron de serie el puerto infrarrojo para el mando a distancia.

La función que realiza el IOP es la de enviar la entrada de los mandos de control o la de los puertos USB al Emotion Engine para que este pueda actualizar el estado del juego apropiadamente.

Estos son los componentes principales de la arquitectura de la PS2, para ampliar más la información en cuanto a la arquitectura de cada uno de los chips de la PS2. Ver [[DOC.01](#)]

A parte de estos componentes la PS2 tiene más hardware, que a continuación detallaremos.

Lector de CD/DVD

Es el principal dispositivo que utiliza la PS2 para cargar los programas que queramos ejecutar en ella. Entre estos programas podemos encontrar desde videojuegos oficiales, discos de música y además para reproducir películas de DVD.

Hoy en día el lector ha perdido importancia con la aparición de aplicaciones capaces de cargar los videojuegos desde un disco duro instalado en la PS2 o con la aparición de cargadores de programas desde PC(loaders) los cuales explicamos en la sección de [Carga de software NO oficial](#).

Los dos puertos para los DualShock

Estos controles son el medio de comunicación del usuario con la PS2, con ellos aparte de jugar, podemos también darle ordenes a la PS2 para poder entre otras cosas reproducir películas de DVD o cambiar algunos aspectos de la configuración.

Las puertos para las tarjetas de memoria (Memory Card)

Estos puertos son utilizados para la conexión de las tarjetas de memoria. Éstas son utilizadas para almacenar los datos de las partidas guardadas, las tarjetas oficiales tienen una capacidad de almacenamiento de 8MB, pero también existen otros

fabricantes no oficiales que han llegado a fabricar tarjetas de memoria de hasta 64MB.

Puertos USB

Estos puertos actualmente están obteniendo mayor importancia porque ahora cada vez más videojuegos utilizan este tipo de conexión. Aparte de ser utilizado para poder conectar accesorios exclusivos para algunos juegos los puertos USB también sirven para conectar disco duros externos, memorias USB, teclados, ratones o micrófonos.

El puerto de expansión Dev9

El puerto Dev9 permite conectar el dispositivo de Ethernet y con él un disco duro ATA, sin embargo la PSTwo para reducir su tamaño se quito este puerto, pero a cambio se le incorporó de serie el puerto de Ethernet. El uso del puerto de expansión Dev9 ha abierto un camino para poder interactuar entre el PC y la PS2.

Otros accesorios

Como comentamos en los puertos USB Sony ha desarrollado videojuegos que utilizan una serie de dispositivos que se comunican mediante estos tipos de puertos:

- *EyeToy* [EYE2004] es una cámara USB capaz de capturar el movimiento del usuario y enviárselo a la PS2 para interactuar con videojuegos específicos ya diseñados para este dispositivo.
- *Los micrófonos* USB utilizados para el *SingStar* [SIN2004] videojuego que simula un karaoke, pero con la especial característica que es capaz del reconocimiento de las entonaciones de la voz.
- *La guitarra* este dispositivo es utilizados para videojuegos en los que simulas tocar una guitarra, entre estos juegos se encuentra el *Guitar Hero* [GUI2005] y todos los que derivan de esta compañía.

2.2. Carga de software Oficial

Una vez explicado el hardware lo siguiente es explicar el software de PS2, el soporte básico de los videojuegos de PS2 son el DVD. PS2 no tiene sistema operativo ni nada parecido a éste, lo único que tiene es un simple menú en el que poder configurar algunos aspectos de la videoconsola y un pequeño reproductor de audio y vídeo.

Debido a este motivo y a que la PS2 ha sido diseñada para cargar software desde la unidad lectora, la carga de software que no sea original de Sony cuesta más trabajo, sobretodo porque para que la PS2 sea capaz de cargar software implementado por el usuario, es necesario grabarlo en CD-ROM y de una forma en concreto que explicaremos en el capítulo de [Entorno de desarrollo para la PS2](#).

Sony para la carga de los videojuegos de PS2 diseño un sistema de seguridad que básicamente consiste en grabar al inicio del CD o DVD unos sectores defectuosos, para que después cuando un usuario intente hacerse una copia de seguridad con su grabadora no le funcione correctamente, ya que las grabadoras caseras en cuanto detectan estos sectores defectuosos se limitan a repararlos en el momento de crear la copia, y luego la PS2 cuando carga la copia busca estos sectores defectuosos y como no los encuentra detecta que no es un juego original.

Como consecuencia de este método de seguridad han surgido unos métodos para poder cargar backups de software oficial de Sony sin la necesidad de que este sea original, estos métodos de carga los podemos agrupar en los modchips y los discos de carga.

Modchip

Es un pequeño dispositivo electrónico utilizado para modificar una videoconsola para añadirle mejoras que no vienen de serie o para eliminar posibles limitaciones a la hora de cargar backups de software oficial y NO oficial.

Discos de arranque

Son unos discos desarrollados para manipular los videojuegos añadiendo a éstos mejoras entre ellas la posibilidad de vidas infinitas, súper velocidad del personaje, salto de niveles, entre otras muchas posibilidades del mismo estilo. Además de esta función también ofrece poder realizar el intercambio de discos (SWAP), para la carga de backups.

Cualquiera de estos dos métodos de carga podemos utilizarlo para el desarrollo del PFC. Las ventajas que vemos del uso del modchip respecto al disco de carga es que el modchip ofrece mayor comodidad ya que no requiere tener que estar haciendo el cambio de discos y además esto aporta mayor seguridad ya que no debemos forzar la bandeja mecánicamente para realizar el cambio.

Por otro lado es importante mencionar que los discos de arranque son un método de carga más económico ya que su precio se aproxima a unos 33€ mientras que la instalación de un modchip acostumbra a costar 100€. También usar discos de arranque son recomendables siempre que no queramos perder la garantía de fábrica, ya que no requieren abrir la PS2 para instalar nada tal y como necesitan los modchips.

2.2.1. Modchips

Modchips hay muchos y para saber cual instalar básicamente nos tenemos que fijar en la calidad y el precio, como en cualquier otro producto que compremos. Aparte de esto dependiendo del modchip que escojamos podremos realizar más o menos cosas con nuestra PS2.

Los tres modchips que hemos escogido para comentar son el Modbo 760, el Matrix Infinity y el DMS 4 Pro.

DMS 4 Pro

Es el más completo de los tres y el más complicado de instalar por su gran número de soldaduras complejas que hay que realizar, por eso también es el más caro de instalar.

Este chip se vende sin programar y una vez instalado nosotros mismos le podemos cargar el software necesario a través de un CD.

DMS4 Pro es la versión profesional de DMS4 Lite. Tiene todas las características de DMS4 Lite añadiendo una memoria flash interna de 2MB para almacenar y ejecutar aplicaciones o sea archivos .ELF sobre todo.

Detecta el tipo de medio insertado (PS1, PS2, CD-R, DVD-R, DVD+R, DVD-RW, DVD9 o DVD+RW) y lo carga automáticamente sin necesidad de métodos de carga o interacción adicional por parte del usuario.

El DMS4 tiene varios tipos posibles de instalación, instalación simplificada (número de cables reducido) 17 cables en V1-V7 PAL/USA o 18 cables en V1-V7 JAP/ASIA . 18 cables en V9-V10 PAL/USA (19 con soporte DVD+/-RW, punto T) o 19 cables en V8,9 y 10 JAP/ASIA (20 con soporte DVD+/-RW, punto T).

En su página oficial podemos encontrar si queremos el diagrama instalación. [[URL.01](#)]

Tiene una alta seguridad ya que el código de DMS4 esta protegido mediante una tecnología de origen militar para protegerlo contra “cloners” y “fakers”.

Es el más compatible con todos los modelos de PS2 de todas las regiones: V1-V10 PAL/USA/JAP.

Su software, fabricado por el equipo de ToxicOs (ver [[URL.02](#)]), se puede actualizar fácilmente desde CD y este software lo que aporta es un menú mucho más completo que el de la PS2 y que nos facilita la posibilidad de utilizar y configurar todas las prestaciones que nos aporta la instalación del DMS4Pro.

Nos permite visualizar películas DVD de cualquier región, fuerza el modo de color y suprime el efecto “verde” en la reproducción de películas DVD con cable RGB.

Es compatible con el modo Dev1, carga aplicaciones y software desde la Memory card de PS2 usando el modo especial DEVolution, también es compatible con el modo Dev2, el cual permite la carga de aplicaciones y software desde el disco duro de PS2. DEVolution mode 2 es el sucesor de DEVolution 1 mode.

Tiene distintas posibilidades de configuración del modo de arranque además de otros parámetros, como la opción de poderlo desactivar fácilmente presionando el botón “cuadrado” en el mando durante el arranque de la PS2.

Matrix Infinity

Es el más equilibrado en cuanto a calidad precio y es el principal rival del DMS4.

Tiene las mismas características que el DMS4 excepto que el modchip Matrix tiene tan solo 512KB de memoria interna.

El software que incorpora este modchip tiene actualizaciones más a menudo respecto del software que tiene el DMS4, pero por otro lado es importante destacar que el

equipo de ToxicOs tarda más en publicar las actualizaciones para el DMS4, para así poder ofrecer muchas más mejoras en su sistema que sus competidores.

Tiene menos soldadura que los otros dos y además tiene mejor soporte por correo electrónico que el DMS 4.

Modbo 760/versiones anteriores

Este modchip es el más económico de los tres que explicamos, es un clon del Matrix pero con mucha menos calidad que éste.

Tiene soporte igual que el DMS4 y el Matrix en lo que se refiere a la carga de aplicaciones y software desde la Memory card o el disco duro de la PS2 (modo Dev1 y Dev2). Estos modos de carga los encontramos en casi todas las versiones disponibles del chip, aunque la versión 760 es la más completa y la más recomendada.

El chip se desactiva al abrir la bandeja después de haber cargado un backup previamente, lo cual hace que este chip sea más incomodo que sus competidores.

En algunas versiones de este chip, permitían que la consola se calentará más de la cuenta, lo cual hacia que se acortara la vida de la videoconsola.

La PS2 con la que realizamos el proyecto tiene instalado un DMS4. La versión del DMS4 es una versión anterior a la versión Pro que acabamos de explicar, con la última versión del software de ToxicOs compatible con este modelo de modchip.

2.2.2. Discos de arranque

Si no queremos tener que manipular la PS2 y mucho menos instalar en ella un chip que manipule las funcionalidades de la videoconsola, entonces debemos optar este tipo de método de carga.

El funcionamiento de este método de carga se basa en el uso de un disco de arranque el cual nos permite realizar un intercambio entre un disco original de PS2 y un backup de PS2 oficial.

Para poder cargar con este método un backup inicialmente hay que arrancar la PS2 con el disco de arranque. Con este disco de carga lo que conseguimos es que siguiendo un proceso, que varía dependiendo del fabricante del disco, la PS2 deje de leer datos del lector en mitad de una carga de un disco de PS2 original.

Se utiliza primero el disco original para que la PS2 lea los sectores defectuosos que explicamos en la sección del análisis del lector de CD/DVD. Una vez ha leído estos sectores defectuosos el software con el que arrancamos para el lector, y es en ese preciso momento cuando, mediante alguno de los métodos existentes *Cogswap* (intercambio en este caso de discos) en la consola, se debe cambiar el disco original de PS2 por el backup que deseamos cargar.

De los distintos tipos de discos de arranque que hay en el mercado el que probamos es el Swap Magic 3 ver [\[URL.03\]](#).

Además de los discos de arranque también necesitamos como comentamos en el párrafo anterior un método para Cogswap y para ello encontramos dos sistemas.

El primero método se basa en quitarle el frontal protector de la bandeja del lector del DVD de la PS2, con la finalidad de así tener espacio suficiente para poder introducir una Slide card, tarjeta

con una ranura ya adaptada para poder abrir de forma mecánica la bandeja, ver [\[URL.04\]](#). Por experiencia propia utilizando este método tienes que realizar el cambio con mucho cuidado, ya que puedes meter la tarjeta antes de tiempo y sin querer enganchar algún componente de esta que este en movimiento.

El segundo método se basa en sustituir la carcasa original de la PS2, por otra la cual en lugar de expulsar la bandeja, se abre una tapa hacia arriba, para así poder realizar el intercambio de discos. El cambiar la carcasa original de Sony por esta especial implica perder la garantía de fábrica.

Para el desarrollo de este PFC, personalmente escogemos la instalación de un modchip porque tenemos que ir ejecutando muchas pruebas y cada vez que realicemos una de ellas deberemos de hacer todo el proceso de Cogswap.

2.2.3. Carga a través de disco duro mediante discos de arranque

Como explicamos en la sección de dispositivos de la PS2, podemos conectar a ésta un disco duro ATA y utilizarlo gracias a software como por ejemplo el HD Advance (ver [\[URL.05\]](#)) para cargar backups de PS2.

El HD Advance son unos discos de arranque que permiten copiar y cargar imágenes al disco duro. También podemos utilizar un backup del propio HD Advance, pero para poder utilizar este backup entonces necesitamos uno de los dos métodos de carga de los anteriores explicados.

2.3. Carga de software NO oficial

Entendemos por software NO oficial o Homebrew, cualquier clase de aplicación de un usuario realizada para una videoconsola, como para este caso la PS2.

Este tipo de software, también puede ser ejecutado utilizando alguno de los dos métodos de carga que explicamos en la sección anterior, pero además existen tres métodos más que nos permite cargar software NO oficial.

Es importante decir que exceptuando el método del uso de un emulador de PS2, el resto de sistemas para cargar software NO oficial requiere del uso de uno de los métodos de carga de backups anteriores.

2.3.1. Disco duro

La idea básica de este sistema de cargar es semejante al método que explicamos para cargar backups en el disco duro utilizando el HD Advance.

La diferencia ahora es que necesitamos utilizar el HDD Dump Tool, este software permite copiar archivos desde el lector de CD/DVD al disco duro de la PS2 para así luego poder ejecutarlo desde este último. Ver [\[URL.06\]](#)

2.3.2. El emulador PCSX2-0.8.1

Otro método estudiado es el uso de un emulador el PCSX2, usar emulador tiene la gran ventaja de que no es necesario tener la PS2 para poder realizar las pruebas de las aplicaciones que compilemos.

Para la instalación del emulador es requisito indispensable descargarnos el PCSX2, la última versión disponible es la 0.9, pero la versión que probamos es la pcsx2-0.8.1, esta versión la descargamos de la página oficial del emulador [[URL.07](#)].

Otra cosa indispensable para poder llevar a cabo la instalación son los plugins necesarios para la configuración del emulador, estos también se pueden descargar la mayoría de la página web oficial.

En la página oficial se encuentra una guía del funcionamiento del emulador y de como configurar cada uno de sus plugins, la cual seguimos para configurar el emulador. Ver [[URL.08](#)]

A continuación comentamos que plugins hemos utilizado para configurar el emulador PCSX2-0.8.1, para el desarrollo del PFC.

Plugin gráfico

El que probamos y utilizamos actualmente para los gráficos es el GSdx9(SSE2)0.8, este plugin soporta el directX 9 y realmente se nota en la velocidad de proceso de los gráficos.

Plugin sonido

En cuanto al plugin de sonido probamos ambas posibilidades tanto el plugin nulo SPU2null v0.3 como el plugin PEOPS SPU2 el cual pese a estar en una versión beta no nos ha causado problemas. En la guía comentan que con el uso del sonido puedes perder 1 o 2 FPS, pero ya que nuestras demos no requieren de grandes cálculos gráficos, no apreciamos ese descenso de velocidad.

Plugin Pad

Tan solo probamos el PADwinKeyb Driver 0.9 , este plugin es necesario para configurar que teclas del teclado emularan los botones del Dualshock de la PS2, este plugin no tiene ninguna dificultad en lo referente a su configuración.

Plugin CDVD

De este plugin probamos el CDVDbin Driver 0.65 con una imagen de un videojuego que teníamos en el PC y funcionó sin problemas. Pero el plugin que es necesario para el proyecto es el CDVDnull Driver 0.6 ya que este es el que nos permite ejecutar los archivos .ELF que generemos en las pruebas.

El resto de plugins no nos han sido necesarios para el proyecto.

Por último nos queda la configuración de la parte más importante del emulador, la BIOS, la cual la extraemos de nuestra PS2 con la ayuda de un Volcador de BIOS. Ver [[URL.09](#)]

Es importante decir el PCSX2 también tiene una versión disponible que corre bajo GNU/Linux, el creador te facilita las fuentes sin ningún problema en la página web oficial, estas fuentes son software libre bajo la licencia GPL.

El único inconveniente de utilizar un emulador como método de carga, es no poder asegurar que funciona totalmente bien ante las aplicaciones que compilemos.

2.3.3. Loaders

Los loaders son aplicaciones diseñadas para cargar software entre dos plataformas, en este caso práctico para cargar software entre el PC y la PS2.

El funcionamiento de este método de carga se basa en la ejecución de dos aplicaciones, una cliente y otra servidor. En este caso la aplicación cliente corre sobre GNU/Linux y la aplicación servidor la arrancamos sobre la PS2.

En la sección del *método de carga que utilizamos para los experimentos* del capítulo siguiente, explicamos con más detalle la configuración de este método.

En la comunidad de PS2DEV [\[URL.10\]](#) encontramos todos los loaders disponibles para cargar software a la PS2. Todos estos loaders los podemos agrupar según el tipo de conexión que utilizan en dos grupos.

Loaders Ethernet

Este tipo de loader requieren que la PS2 tenga conexión Ethernet.

Dentro de este grupo encontramos por la parte del cliente el *pksh* y el *PS2client*. Ambos loaders tiene un funcionamiento similar, que se basa en abrir una consola de comandos para poder ejecutar instrucciones sobre la PS2.

Por el lado de la PS2 tenemos el PS2link, esta aplicación se ejecuta en el momento de arrancar la PS2 y esta a la espera de recibir instrucciones desde el cliente instalado.

Loaders USB

En esta agrupación se encuentran todos los loaders que utilizan una conexión USB para la carga de software.

Para poder realizar la conexión con estos loaders necesitamos disponer de un cable USB que tenga el chip PL2301, este tipo de cables lo podemos comprar en cualquier tienda de electrónica especializada.

El cliente que hay disponible para este tipo de loaders es el *npsh*. Este loader tiene de igual manera que los clientes Ethernet una consola por la que enviamos instrucciones a la aplicación servidor.

Como aplicación servidor encontramos el *Naplink* el cual debemos usar a la hora de arrancar para así poder realizar la conexión con el PC.

De este método de carga opinamos que para llevar a cabo los experimentos que debemos de hacer es el más práctico.

2.4. Entornos de desarrollo para PS2

Podríamos decir qué entendemos por entorno de desarrollo:

Un conjunto de herramientas y aplicaciones utilizadas para poder programar con ellas, pueden dedicarse en exclusiva a un sólo lenguaje de programación o bien, usarse para varios.

Durante la fase de búsqueda de los diversos entornos que existen para desarrollar software para PS2, encontramos principalmente tres, de estos explicamos con más detalles los que se adapten a los requisitos no funcionales del PFC.

2.4.1. Kit oficial de desarrollo de Sony

Las compañías de desarrollo de videojuegos, que quieren desarrollar para la PS2 de manera oficial, Sony les ofrece un kit de desarrollo el cual consta de dos equipos.

- El equipo para realizar el desarrollo es más grande que una PS2 normal, tiene más RAM, un disco duro y ethernet integrados.
- El equipo de pruebas es igual que la PS2 normal, pero con la diferencia de que puede leer backups, sin la necesidad de ningún sistema de arranque.

Este equipo esta pensado para que puedan trabajar un grupo de dos personas a la vez, desarrollando y probando lo que implementan.

Este método de desarrollo no esta pensado con la idea de que una pequeña empresa pueda diseñar videojuegos para PS2, ya que necesitaría de una cantidad razonable para la inversión de dichos entornos. Estos entornos cuestan aproximadamente unos 20.000\$ cada kit.

Debido a esto y a razones legales ningún usuario sencillo puede diseñar con dicho entorno, para desarrollar sus aplicaciones para PS2.

2.4.2. Kit de GNU/Linux para PS2

Otro entorno de desarrollo distribuido por Sony que desde hace unos años se encuentra en el mercado es el kit de desarrollo GNU/Linux basado en Debian GNU/Linux [DEB1997], para que los usuarios de PS2 pudieran desarrollar sus propias aplicaciones para sus consolas.

El kit está compuesto de dos discos (Figura 2.1). El primero contiene la documentación de Sony y todo lo necesario para la ejecución del GNU/Linux, mientras que en el segundo viene toda una serie de aplicaciones para poder instalar en él.

Este kit también tiene componentes y periféricos para PS2, entre ellos encontramos un teclado y un ratón USB, un adaptador de red, un disco duro de 40 GB y un adaptador VGA para poder conectar la consola al monitor.

Sony con la comercialización de este kit intenta ofrecer al usuario la posibilidad de desarrollar sus propias aplicaciones para PS2, el problema de este entorno de desarrollo es que tiene un par limitaciones bastante importantes a la hora de decidirse desarrollar en él.



Figura 2.1: Kit oficial de GNU/Linux para PS2

La primera limitación es que Sony ofrece al usuario tan solo los archivos binarios de los driver de entrada y salida, y la segunda es que todo lo que se desarrolle con este entorno, solo servirá en otras PS2 que tengan también el kit de GNU/Linux instalado.

Aun así en el momento de investigar los diversos entornos de desarrollo para PS2, una opción interesante resulta tener la posibilidad de instalar una distribución GNU/Linux en la PS2 para directamente desarrollar desde ella.

Además del kit oficial de Linux que distribuye Sony, existe una distribución de Linux llamada BlackRhino basada en Debian.

A continuación analizamos la instalación de la distribución BlackRhino la cual realizamos a partir de un manual que encontramos en internet [[URL.11](#)].

Es necesario para poder instalar BlackRhino que tengamos algún método que nos ofrezca la posibilidad de arrancar la PS2 en modo Dev1, ver sección [Modchips](#) de este mismo capítulo. En el manual utilizan el primer disco del kit de Linux para arrancar en Dev1.

Para poder copiar la imagen de BlackRhino que ocupa 21MB, es necesario trocear la imagen ya que las tarjetas de memoria son de 8Mb. Los trozos no pueden superar los 3Mb porque aparte de el trozo de la imagen también es necesario añadir a la tarjeta un kernel para que arranque un Linux y así poder montar la tarjeta para copiar el trozo de BlackRhino al disco.

El proceso no es complicado pero si es lento, como disponemos de dos tarjetas de memoria, el proceso se agiliza ya que podemos copiar tres trozos (uno de la primera tarjeta y dos trozos de la segunda).

Cosas importantes que hay que tener en cuenta ante todo, lo más importante es que a la hora de trocear la imagen la aplicación que usemos no añada nada de código a los trozos. Otra cosa importante a tener en cuenta es que antes de empezar copiemos al PC todas las partidas guardadas que tengamos en la memory card, ya que si no corremos el riesgo de perderlas.

Una vez copiados todos los trozos en el disco duro de la PS2 queda volver a unirlos y descomprimir, el BlackRhino.

En el manual podremos ver más detallado cada uno de los pasos que explicamos.

El único error que encontramos en este manual está en el link que hace referencia a la página donde podemos descargar la imagen del BlackRhino, el cual es erróneo. La dirección correcta donde encontrar todo lo referente la podemos ver aquí [[URL.12](#)].

La limitación de que cualquier cosa que implementemos con este entorno tan solo pueda ser utilizado por usuarios que tenga instalado Linux en la PS2, no nos atrajo porque queríamos aportar algo que pudiera utilizar cualquier tipo de usuario. Además de esto también encontramos otro inconveniente a la hora de trabajar con en este entorno de desarrollo, la dificultad de poder leer los comandos por la pantalla del televisor.

2.4.3. Programación RAW

La programación raw es un método de desarrollo de aplicaciones para PS2 mediante el uso de compiladores cruzados y sin usar ni el kit oficial de Sony ni el kit de Linux.

En la comunidad OpenSource podemos encontrar el PS2DEV entorno de programación RAW, este entorno tiene todo tipo de herramientas y librerías para poder desarrollar diversas aplicaciones para PS2. Cualquiera de estas herramientas la podemos obtener de su repositorio oficial, ver [[URL.13](#)].

Las ventajas que encontramos de este entorno de desarrollo para trabajar eran:

- Al estar desarrollado en OpenSource está en continuo desarrollo y esto facilita que esté actualizado y siempre podamos obtener la última versión estable.
- El PS2Dev tiene un port de las librerías multiplataforma SDL, las cuales consideramos interesantes para estudiar la portabilidad de juegos de PC.
- Otra razón que nos resulta interesante es que el PS2Dev tiene unas herramientas que facilitan el proceso de instalación de las herramientas básicas necesarias del entorno, para después poder ir instalando las librerías o aplicaciones que necesitemos para las pruebas.

2.5. Compilación cruzada

Un entorno de desarrollo dijimos que era un conjunto de herramientas y aplicaciones para poder desarrollar programas con ellas, siguiendo esta definición podemos decir que un entorno de compilación cruzada son las herramientas para desarrollar aplicaciones para una plataforma la cual no es desde la que estamos programando, por ejemplo sería crear una aplicación en un sistema POSIX para después ejecutarlo en un sistema Microsoft, ver Figura 2.2.

Hay varios casos en los cuales es necesario utilizar compilación cruzada:

- Cuando vamos a diseñar aplicaciones para una plataforma la cual no dispone de un sistema operativo en el que podamos configurar las herramientas necesarias para desarrollar desde el propio sistema.
- Que el desarrollo en cuestión desde la máquina a desarrollar sea una tarea muy incomoda o lenta, por ejemplo programar una aplicación para una PDA.
- Que la máquina para la que vamos desarrollar todavía esté en fabricación, un ejemplo claro es la creación del software necesario para una centralita de un coche el cual todavía esta en el proceso de montaje.

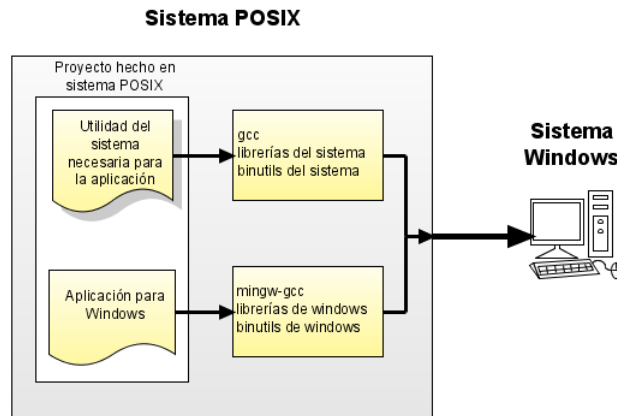


Figura 2.2: Diagrama de entorno de compilación cruzada.

Pero utilizar este tipo de entornos tiene sus inconvenientes, los largos procesos de instalación y configuración que hay que realizar para dejarlos a punto. Y la dificultad añadida a la hora de depurar o realizar alguna tarea que necesite ser ejecutada desde la plataforma de destino.

Para poder tener un entorno de compilación cruzada necesitamos básicamente tres elementos o componentes:

- **Compilador:** compilador de C básico en este caso práctico.
- **Librería estándar de C:** encargadas de las llamadas al sistema mediante APIs.
- **Binutils:** conjunto de programas para compilación, enlazado, ensamblado y depuración de código.

Configurar este tipo de entornos es complicado porque como estos elementos dependen entre ellos y existe diversas versiones de cada una de estas herramientas, es difícil encontrar la configuración de versiones que sea compatible entre ellas. En el capítulo de [Entornos de desarrollo para PS2](#) explicamos las versiones que componen el entorno de desarrollo que configuramos para los experimentos de portabilidad.

De igual modo que la plataforma de desarrollo necesita estas herramientas, también tenemos que configurar las mismas herramientas para la plataforma de destino.

Como la PS2 posee dos microprocesadores internamente, el entorno de programación RAW es algo más complejo al tener que configurarse un entorno cruzado para el EE y otro para el IOP. Ambos compiladores son versiones del compilador gcc del lenguaje de programación C. Además también son necesarios sus correspondientes ensambladores y el resto de herramientas.

2.5.1. Canadian Cross

Hay ocasiones en las que en el entorno de compilación cruzada interviene una tercera plataforma, cuando encontramos esta situación estamos ante una variante de la compilación cruzada

conocida como Canadian Cross.

Es una técnica utilizada para crear compiladores cruzados para otras máquinas. A partir del diagrama de la Figura 2.3 explicaremos el concepto principal de la Canadian cross.

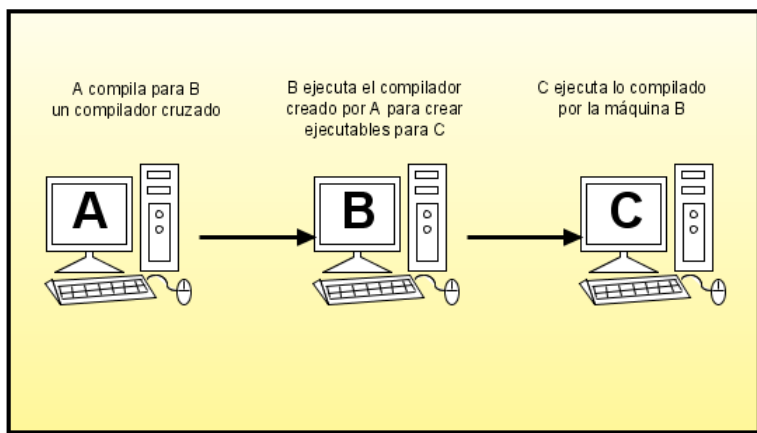


Figura 2.3: Diagrama de entorno canadian cross.

Lo que este gráfico nos intenta representar es que en el proceso de desarrollo de una aplicación para la *máquina C* interviene primero una *máquina A* la cual tras construir su propio compilador, por ejemplo un gcc, y después con éste desarrolla un entorno de compilación cruzado para que desde la *máquina B* desarrollemos la aplicación que queríamos para la *máquina C*.

Al verse implicadas tantas máquinas y versiones de compiladores este proceso es bastante más laborioso y delicado, y cualquier pequeño error que podamos tener en alguno de los pasos de su configuración puede ser después complicado de encontrarlo.

Analizando el entorno de programación RAW que explicamos en el apartado anterior, podemos confirmar que se trata de un caso de Canadian Cross.

Siguiendo la figura de arriba tenemos que el PS2DEV ha sido desarrollado por una *máquina A* que no conocemos, por otro lado la *máquina B* es el PC con Linux desde el que realizamos este PFC y en el cual hemos instalado el PS2DEV, por último la *máquina C* es la plataforma de destino de las aplicaciones que implementemos, en nuestro caso es la PS2.

2.6. Autotools

GNU tiene desarrollados unos paquetes de herramientas llamados **autotools** [AUT2003].

Creemos interesante utilizar las autotools porque permiten portar el código de aplicaciones dejando a un lado el problema de las versiones de las herramientas. Las autotools a partir del script *configure* se encargan de verificar la versión de las herramientas necesarias para la compilación y también se encarga de configurar todas las referencias a estas herramientas.

Estudiando la documentación de las autotools encontramos que estas herramientas ofrecen soporte para realizar compilación cruzada, ver [URL.14]. Como tenemos el objetivo de realizar

una portabilidad, resulta interesante estudiar como portar aplicaciones utilizando estas herramientas.

Las autotools son unas herramientas de UNIX creadas para el desarrollo de aplicaciones, en estas herramientas están incluidas el `aclocal`, el `autoheader`, el GNU `autoconf`, el GNU `automake`, el `configure` y el `make`. A continuación explicaremos para que se utilizan cada una de estas herramientas.

aclocal

Es la herramienta que genera de forma automática el archivo `aclocal.m4`, que es el empleado por `autoconf` para buscar las macros de `automake` dentro de los archivos con prefijo `AM'_` del directorio donde lo invoquemos y en último lugar busca también dentro del fichero `configure.in`.

autoheader

Crea una plantilla con un conjunto de directivas `#define` que puede emplearse desde los programas en C. Para definir el nombre de los archivos de cabecera que queremos que se incluyan en el código para poder utilizarlos, debemos usar la macro `AC_CONFIG_HEADERS`. Por ejemplo si quisiéramos incluir el archivo `SDL.h`, bastaría con declarar en el archivo `configure.in` la macro de la siguiente manera:

```
AC_CONFIG_HEADERS (sd1.h)
```

Entonces `autoheader` al interpretar la macro creará el archivo `config.h.in`, y este en un proceso posterior pasará a ser el archivo `config.h`.

autoconf

Es la herramienta que genera el script encargado de comprobar todos los requisitos de el sistema para realizar la construcción de una aplicación. Esta herramienta nos permite construir paquetes de programas portables y también nos facilita la instalación y la desinstalación de las aplicaciones que creamos con ella. `Autoconf` genera el archivo `configure` a partir de las macros definidas en los fichero `configure.in` y en el fichero `aclocal.m4` si este último existe.

automake

Ayuda a crear archivos `Makefile` portables. A partir del archivo `Makefile.am` genera un archivo `Makefile.in`, es capaz de generarlos para varias aplicaciones a la vez aunque estén ubicadas en distintos paths siempre y cuando incorporemos en cada directorio un fichero `Makefile.am`.

2.7. Simple DirectMedia Layer (SDL)

Aparte de las *autotools* también puede realizarse la portabilidad de una aplicación tan solo usando los compiladores cruzados necesarios y utilizando un lenguaje multiplataforma y las librerías de este.

Investigando encontramos en el PS2DEV un port de las SDL. SDL ha sido diseñado pensando en que sea fácil de portar. Esto implica que si un programa usa SDL será más fácil portarlo, pero en ningún caso es una *“herramienta para realizar la portabilidad”*.

Al tener disponible un port de SDL para PS2, decidimos portar un videojuego hecho en SDL, utilizando estas librerías gráficas.

Las API de SDL proporciona funciones para realizar dibujos 2D, manejar música y efectos de sonido, y gestionar carga de imágenes.

Además de las propias librerías básicas de la API, SDL también tiene otras librerías complementarias disponibles:

- **SDL_image**
es la encargada de cargar imágenes de distintos formatos: jpg, png, etc.
- **SDL_mixer**
a diferencia de las SDL_image esta carga formatos de sonido: ogg, mp3, wav, etc.
- **SDL_ttf**
estas se encarga de la gestión de los tipos de fuentes.
- **SDL_gfx**
control de framerate, escalar/rotar imágenes, filtros de imagen MMX, dibujo de primitivas gráficas.
- **SDL_net**
se encarga de las comunicaciones de redes.

Si queremos dotar al videojuego de efectos 3D o aceleraciones 2D, con las SDL no podemos conseguir esto, pero podemos compaginarlas con las OpenGL, para conseguir tal propósito.

La portabilidad que ofrece SDL a otras plataformas, hace que esta API sea la candidata más apta para llevar acabo este proyecto. Podemos encontrar mucha documentación de estas librerías en su página oficial, ver [URL.15], pero por la parte del port de PS2 esta documentación es nula, y esto es un inconveniente.

Capítulo 3

Entorno de desarrollo para la PS2

En este capítulo explicamos todo lo necesario para poder configurar el entorno de programación RAW que escogimos para realizar la portabilidad.

En un principio explicamos el sistema operativo y las herramientas básicas que tiene el entorno de compilación cruzado con el que realizamos la portabilidad. Seguidamente ya profundizamos en las herramientas más específicas para este caso práctico, como el entorno de desarrollo PS2DEV y el método de carga escogido que es el loader ethernet Pukklink Shell(pksh).

También explicamos la configuración del port de las librerías SDL que hemos escogido para realizar la portabilidad del videojuego.

3.1. Sistema operativo

Los requisitos no funcionales del proyecto, exigen que se desarrolle todo el proyecto con software libre, por esta razón escogemos como sistema operativo una distribución de GNU/Linux.

Por la facilidad de instalación y configuración escogemos la distribución Ubuntu 7.04 Feisty Fawn, pero podemos escoger cualquier otra distribución mientras sea de un sistema POSIX.

3.2. Software para desarrollar

Además de Ubuntu, como explicamos en el capítulo de análisis en la sección [Compilación cruzada](#) también son necesarias unas herramientas básicas para desarrollar.

La mejor opción encontrada es el compilador *gcc* de GNU, ya que ofrece la posibilidad de realizar compilación cruzada. La versión de gcc que instalamos, es la gcc 4.1.2 [[URL.16](#)].

```
apt-get install gcc-4.12
```

Para poder construir el compilador cruzado para PS2, también necesitamos las librerías básicas de C (*libc-dev*), las *build-essential* y las *binutils*, en el caso de las *binutils* usamos la versión 2.14.

Para instalar estas librerías, siendo usuario **root**, escribimos los siguientes comandos:

```
apt-get install binutils  
apt-get install libc-dev  
apt-get install build-essential
```

Después de instalar el compilador y estas tres librerías ya tenemos preparado el sistema operativo y las herramientas necesarias para instalar el entorno de programación RAW (PS2DEV).

3.3. Entorno de programación RAW (PS2DEV)

El entorno de programación RAW más completo que encontramos es el PS2DEV. Además este entorno ofrece una instalación muy práctica mediante el uso de las herramientas toolchain de GNU (colección de herramientas desarrolladas por GNU para facilitar el desarrollo de aplicaciones) [[URL.21](#)].

El primer paso que debemos realizar es obtener las fuentes del PS2DEV. Para obtener una copia de las fuentes del PS2DEV, podemos descargarlas de su repositorio oficial, ver [[URL.13](#)]. Con el siguiente comando descargamos una copia del PS2DEV:

```
svn co svn:/svn.ps2dev.org/ps2/trunk/
```

Otra cosa necesaria para la instalación del PS2DEV es declarar las variables de sesión *PS2DEV*, *PS2SDK* y modificar la variable *PATH*.

Como estas variables también las utilizaremos para los experimentos, para no tener que declararlas cada vez que iniciemos una sesión, las añadiremos a los archivos *.bashrc* del usuario con permisos **root** que utilizamos. Para declarar las variables tan solo es necesario añadir las siguientes líneas al final del fichero:

```
export PS2DEV=/usr/local/ps2dev  
export PATH=$PATH:$PS2DEV/bin  
export PATH=$PATH:$PS2DEV/ee/bin  
export PATH=$PATH:$PS2DEV/iop/bin  
export PATH=$PATH:$PS2DEV/dvp/bin  
export PS2SDK=$PS2DEV/ps2sdk  
export PATH=$PATH:$PS2SDK/bin
```

Si el usuario que vamos a utilizar no tiene permisos de *root* deberemos cambiar el path de la variable PS2DEV por una dirección en la que tengamos permisos totales.

Ahora el siguiente paso que debemos hacer es ejecutar las toolchain, para ello primero debemos darle permiso de ejecución al fichero *toolchain.sh*, mediante el comando `chmod`.

```
cd trunk/ps2toolchain/

chmod 777 toolchain.sh
```

Y una vez tiene permisos de ejecución tan solo queda ejecutar el script para la instalación.

```
./toolchain
```

El proceso de instalación del PS2DEV tarda aproximadamente veinte minutos dependiendo del PC.

3.4. Librerías SDL

Para la portabilidad que vamos a realizar es necesario también la instalación del port las librerías SDL.

Para la instalación del port de SDL de PS2 debemos primero instalar las librerías **libtiff** y **gsKit**, estas librerías básicas sirven para el tratamiento de imágenes. Disponemos de las fuentes de estas librerías dentro del propio PS2DEV.

Instalación de la libtiff

Para que no genere un mensaje de error de compilación al intentar instalar esta librería es necesario editar el archivo *Makefile* y añadir la siguiente línea, que encontramos que faltaba.

```
LIBTIFF = .
```

También por comodidad y para que sea un proceso más automatizado la instalación de estas librerías, podemos añadir al fichero *Makefile* también las siguientes líneas.

```
install:

    cp \*.a $(PS2DEV)/ee/lib

    cp \*.h $(PS2DEV)/ee/include
```

Con estas líneas en el momento de hacer *make install* ya make copiará cada cosa en el lugar que le toca.

Una vez modificado el *Makefile* tan solo hay que ejecutar.

```
make

make install
```

Instalación de la gsKit

Para instalar esta librería, antes de nada tenemos que modificar el fichero *Makefile.pref* y comentar la línea que declara la variable PS2DEV, ya que el comando *make* sustituiría nuestra variable de sesión PS2DEV que tenemos declarada correctamente por esta que no tiene porque estar apuntando al path correspondiente del entorno.

Después igual que la libtiff solo hay que ejecutar.

```
make

make install
```

Después de haber instalado estas dos librerías ya podemos instalar la API de SDL, la cual se encuentra dentro del directorio *ps2sdk-ports*.

```
cd trunk/ps2sdk-ports/sdl/

make

make install
```

Llegados a este punto tenemos configurado el entorno de desarrollo de PS2 para poder programar aplicaciones en SDL. Es importante saber que dentro del directorio *ps2sdk-ports* también están disponibles todas las librerías de SDL que comentamos en el capítulo anterior en la sección dedicada a las [Simple DirectMedia Layer \(SDL\)](#).

3.5. El método de carga de software NO oficial

Por último para ejecutar todos los test que realizamos es necesario un método de carga de software NO oficial, ver sección [Carga de software NO oficial](#) para más información.

Escogemos como método de carga un *loader ethernet* ya que encontramos que es el más práctico a la hora de trabajar, una vez lo tenemos configurado.

El software cliente que escogemos para Linux es el **Pukklink Shell (pksh)** [[URL.17](#)]. El pksh es una aplicación que se conecta de manera remota a una aplicación servidor que este arrancada en la PS2.

Para la instalación del **pksh** primero debemos instalar la librería *libreadline*. Y una vez cubrimos esa dependencia ya podemos instalarlo.

```
apt-get install libreadline

cd trunk/pksh

make pksh
```

Ahora ya tenemos instalado el loader cliente en Linux, pero también necesitamos un loader servidor que correrá del lado de la PS2.

Escogemos el ps2link v1.46 [URL.18] como aplicación servidor para la conexión por parte de la PS2. Como tenemos un método para poder cargar backups de CD/DVD no firmados por Sony, decidimos grabar el ps2link en un CD para arrancarlo desde ese dispositivo.

Para realizar la creación del backup seguimos el manual creado por el miembro Suloku y titulado con “Mini tutorial: Crear un boot cd” [URL.20], que encontramos en el foro de la página web ElOtroLado.net [URL.19].

En este manual ya explican como crear la imagen ISO para después grabarla al CD. Pero explicaremos que ficheros son necesarios para grabar el ps2link y la estructura básica que debe de tener la ISO.

system.cnf

Es el fichero autorun del cd, esto quiere decir que editando este fichero podemos elegir entre otras cosas qué aplicación de las que tiene el cd, será la que queremos que arranque en el momento que la PS2 cargue el disco. El fichero system.cnf internamente tiene que cumplir el siguiente formato.

```
BOOT2= cdrom0:PS2LINK.ELF; 1
VER= 1.0
VMODE= PAL
```

Con la línea BOOT2 indicamos que la aplicación que queremos que autoarranque se llama ps2link.elf y que se encuentra dentro del CD-ROM. Es muy importante que el nombre del ejecutable este completamente en mayúsculas en el fichero system.cnf, aunque en el CD-ROM no este en mayúsculas.

La línea que pone VER, es para indicar el número de la versión del CD-ROM grabado.

Y la línea de VMODE es la que se encarga de indicar el formato de vídeo en el que queremos ejecutar la aplicación, el otro formato que podemos escoger es el NTSC declarar estos formatos tan solo son necesarios para decir la región.

***.ELF**

Estos archivos son los ejecutables para la PS2. Entre ellos debe de encontrarse el ps2link.elf, que ejecutará la PS2 en el momento que arranque desde el cd. El nombre de los programas ejecutables no pueden superar el 8+3 número de caracteres, porque sino la PS2 no los reconocerá, ver [URL.19].

En un mismo CD puede haber más de un ejecutable, pero solo se ejecutará el que esté en el system.cnf, y los otros podemos ejecutarlos después por nuestra cuenta ya que los tenemos en el CD-ROM también.

DUMMY

Este fichero resulta indiferente lo que contenga, pero lo que si que importa es que lo incorporemos en nuestra ISO del boot cd. El fichero DUMMY puede ser cualquier

fichero que tengáis de más de 100Mb, renombrado por supuesto. Su uso es para rellenar el cd, para que este pese más de lo que pesaría con el ps2link u otras aplicaciones de tan pequeño tamaño. Si el CD-ROM no alcanza un peso igual o superior a 100Mb, la PS2 puede no ser capaz de ejecutarlo.

En el caso de la grabación de ps2link además de estos ficheros comunes también es necesario añadir el fichero de configuración de red **IPCONFIG.DAT** ya que con este fichero le indicamos los parámetros de configuración para configurar la tarjeta de ethernet. También será necesario añadir unas librerías específicas para PS2 las cuales requiere ps2link para su funcionamiento, entre ellas se encuentran: **PS2IP.IRX**, **PS2LINK.IRX**, **IOMANX.IRX** y **PS2DEV9.IRX**.

La estructura que tiene el fichero de configuración IPCONFIG.DAT es muy simple y necesita que le indiquemos la dirección ip, la mascara de red y la puerta de enlace, de la siguiente manera:

```
192.168.0.11 255.255.255.0 192.168.0.1
```

Una vez ya tenemos grabado el ps2link en un CD tan solo nos queda configurar la red de GNU/Linux por ejemplo con las siguientes direcciones:

```
Dirección ip: 192.168.0.12
```

```
Mascara de subred: 255.255.255.0
```

```
Puerta de enlace: 192.168.0.1
```

Y ahora tan solo arrancando la PS2 con el CD grabado del ps2link y ejecutamos el *pksh* en el Linux, ya tendremos la PS2 y el PC conectados.

El *pksh* dispone de una serie de comandos, pero principalmente el más utilizado es **eeexec** **<file>**, con este comando ejecutamos remotamente en la PS2 las pruebas *.ELF* que realicemos.

Capítulo 4

Portando un juego a la PS2

En este capítulo explicamos los distintos experimentos realizados durante el estudio realizado para portar una aplicación desde un PC, con Ubuntu 7.04, a una PS2. Para realizar la portabilidad utilizamos la metodología de la Compilación Cruzada, la cual ya explicamos en que consistía en la sección [Compilación cruzada](#) del capítulo de Análisis.

A lo largo de este capítulo mostramos todos los experimentos realizados, explicamos el objetivo deseado de cada uno de ellos y analizamos a que conclusiones llegamos después de su ejecución.

Todos los experimentos que realizamos los agrupamos por secciones. En cada sección, estos experimentos tienen en común que están probando la misma funcionalidad. Dentro de cada sección los presentamos en orden de menor a mayor dificultad.

Para realizar todos estos experimentos, lo primero que tenemos que hacer es configurar todo el entorno de desarrollo como hemos explicado en el capítulo [Entorno de desarrollo para la PS2](#).

4.1. Tests básicos del entorno PS2DEV

Para comprobar el entorno de programación RAW (PS2DEV) que instalamos y los distintos dispositivos de PS2 con los que podemos interactuar, realizamos unos test básicos que se encuentran disponibles en las fuentes del PS2DEV.

Entre estas aplicaciones encontramos *testtimer*, *testcdrom*, *testendia*, *testerror*, *testjoystick*, *testver*, *checkkeys* y un archivo Makefile encargado de compilar todos estos test. Todos estos test no están preparados para ser compilados con las autotools.

Una vez ejecutamos *make* comprobamos que todos estos test compilan. El siguiente paso es ejecutar cada uno de ellos contra la PS2 utilizando el *pksh*, para ver el resultado que obtenemos.

Después de probar cada uno los test, observamos que funcionan todos correctamente exceptuando el test *checkkeys*. Este ejemplo captura las pulsaciones del teclado y el ratón que tenemos conectado a la PS2, pero al ejecutarlo en la PS2 no captura ninguna acción que realizemos. Para asegurar que el código fuente de *checkkeys* es correcto, compilamos *checkkeys* en el GNU/Linux y al ejecutarlo comprobamos que en Linux funciona correctamente.

La conclusión que obtenemos de la ejecución de estos test es que la entrada por teclado y ratón

no las controla bien el entorno de desarrollo, igualmente decidimos seguir adelante porque el DualShock si que lo detecta correctamente para capturar datos de entrada a través de él.

También decidimos que a partir de estos experimentos básicos, vamos a diseñar una estructura de directorios y de archivos *Makefile* que agilicen los próximos experimentos que realicemos.

El Makefile genérico que diseñamos a partir del Makefile de los test básicos tiene el siguiente formato:

```

BINS = cuadrado.elf

OBJS =
EE_INCS = -I../include
EE_LIBS = -L. -lc -L$(PS2DEV)/gsKit/lib -L../lib -lsdl -lcdvd

all: $(BINS)

install:

%.elf : %.o $(PS2SDK)/ee/startup/crt0.o
        $(EE_CC) -mno-crt0 -T$(PS2SDK)/ee/startup/linkfile $(EE_LDFLAGS) \
        -o $@ $(PS2SDK)/ee/startup/crt0.o $< $(EE_LIBS)

clean:
        rm -f $(BINS) $(OBJS)

include ../../Makefile.pref
include ../../Makefile.eeglobal

```

En este Makefile tan solo tenemos que cambiar el nombre del binario que vamos a compilar (el archivo .elf de la variable BINS). Y también tenemos que indicarle las librerías e includes que necesite el propio ejemplo a compilar, mediante las variables EE_INCS y EE_LIBS.

Tenemos que tener también en cuenta que los dos últimos includes del final, siempre deben de tener la ruta adecuada según cada ejemplo, para que make los incluya sin problemas, ya que estos tienen todo el resto de librerías genéricas y los diversos flags que necesita make para compilar los ejemplos.

Además del Makefile que creamos también realizamos una copia de los archivos Makefile.pref y Makefile.eeglobal a la raíz del directorio de pruebas. También es necesario al fichero Makefile.eeglobal indicarle donde están las librerías y los archivos de cabecera del entorno de desarrollo. Para ello lo modificamos cambiando las siguientes líneas:

```

# Include directories
EE_INCS := -I$(PS2SDK)/ee/include -I$(PS2SDK)/common/include -I. \
        $(EE_INCS)

```

por:

```
# Include directories
EE_INCS := -I$(PS2SDK)/ee/include -I$(PS2SDK)/common/include -I. \
          $(EE_INCS) -I$(PS2SDK)/ports/include/SDL
```

Y también:

```
# Linker flags
EE_LDFLAGS := -L$(PS2SDK)/ee/lib $(EE_LDFLAGS)
```

por:

```
# Linker flags
EE_LDFLAGS := -L$(PS2SDK)/ee/lib -L$(PS2SDK)/ports/lib $(EE_LDFLAGS)
```

Después de estos pasos denenmos la siguiente estructura:

```
tests_PS2

    Makefile.pref

    Makefile.eeglobal (modificado con para el entorno)

    tests_basicos

        checkkeys.c

        testerrors.c

        ... (el resto de test básicos)

        Makefile (el genérico con solo los BINS a compilar)

    tests_portabilidad

        cuadrado.c

        Makefile (el genérico indicando el BIN a compilar)
```

A partir de este momento todos los test que realizamos los adaptamos a esta estructura de Makefiles.

4.2. Test del port de SDL con un proyecto PS2 en SDL

Este test también lo realizamos para probar aspectos de la configuración, pero éste prueba el port de SDL para PS2.

Buscando por la red conseguimos encontrar el Super Mario Wars(SMW) [SMW2006] un juego hecho en SDL del famoso fontanero de la Nintendo [NIN1889].

Si nos fijamos en los requisitos de este videojuego vemos que son necesarias tener instaladas la `SDL_image` y la `SDL_mixer`, estas librerías las tenemos en el mismo directorio donde se encuentran las fuentes del port de las SDL.

Así que para poder compilar el SMW primero instalaremos estas librerías.

Instalación de las `SDL_image`

Para poder instalar esta librería primero es necesario instalar las `libjpeg` y las `libpng`, las `libpng` a su vez requieren de las `zlib`. Una vez instaladas todas estas librerías, ya podemos instalar las librerías `SDL_image`.

```
cd ../libjpeg/  
  
make  
  
make install  
  
cd ../zlib/  
  
make  
  
make install  
  
cd ../libpng/  
  
make  
  
make install  
  
cd ../sdlimage/  
  
make  
  
make install
```

Con esta serie de comandos ejecutados desde el directorio `./trunk/ps2sdk-ports/sdl` ya tenemos instaladas la `SDL_image` y todas sus dependencias.

Instalación de las `SDL_mixer`

Esta librería no dependen de otras libs, así que solamente tenemos que ejecutar:

```
cd ../sdlmixer/  
  
make
```

```
make install
```

Después de estos pasos ya tenemos en el entorno configurado con estas dos librerías que necesitamos para compilar el SMW.

Ahora ya podemos proceder a la instalación del SMW pero antes hay que corregir unos errores léxicos.

En el archivo *music_player.h* hay un problema de mayúsculas y minúsculas, error debido seguramente por programar en una plataforma que no es sensible a las mayúsculas y minúsculas.

La solución se remite a substituir la siguientes líneas.

```
#include <SDL_Mixer.h>
```

por:

```
#include <SDL_mixer.h>
```

El otro error que encontramos esta en el archivo *Makefile*, este incluye una librería que el compilador dice que no existe, debido a que está mal escrita.

```
EE_LIBS += -L$(PS2SDK)/ports/lib -lsdlmain -lSDL_image -lsdlmixer \
          -lsdl -ljpeg -lmc -ldebug
```

La solución es substituir *-ljpeg* por *-ljpeg*.

Una vez corregidos estos pequeños errores tan solo quedaba compilar ejecutando *make*. Éste genera un archivo llamado *smw.elf* el cual es el ejecutable resultante de la compilación.

Finalmente podemos comprobar a través del pksh, que funciona el SMW en la PS2 sin problema alguno.

Después de la ejecución de este test tenemos una ligera seguridad de que el entorno esta bien configurado, al menos para este ejemplo ha funcionado sin problemas.

4.3. Test de portabilidad básicos

Una vez testado el funcionamiento de las SDL en PS2 los siguientes test que ejecutamos se basan en una serie de ejemplos sencillos de portabilidad de SDL, que testean básicamente la librería *SDL_image*.

4.4. Cuadrado

Este experimento dibuja un cuadrado pintado píxel por píxel por la pantalla. Con este experimento comprobamos el funcionamiento de la librería *SDL_image*. También servirá para comprobar que la estructura de Makefiles que tenemos diseñada funcionaba correctamente.

Para adaptar Cuadrado a la estructura que diseñamos, el primer paso que realizamos es descomprimir sus fuentes dentro del directorio de *tests_portabilidad*. El siguiente paso es copiar el Makefile genérico y modificar las variables BINS, EE_INCL Y EE_LIBS. También es necesario modificar los includes de los archivos *Makefile.eeglobal* y *Makefile.pref*.

Por otro lado dentro del código de *cuadrado.c*, tenemos que modificar la definición de la función *main* ya que si no lo hacemos devuelve un error de **definición de múltiples main**, debido a que el port de la librería de SDL tiene ya una función *main* propia declarada. En la sección [Test de portabilidad de un proyecto SDL que utiliza autotools](#), de este mismo capítulo esta más detallado este error.

Una vez tenemos todo configurado, compilamos ejecutando *make*. El resultado que obtenemos es que Cuadrado funciona, y por la pantalla de la televisión vemos como la PS2 dibujaba un cuadrado.

Este resultado confirma que podemos portar por el momento aplicaciones muy básicas con el entorno de desarrollo que tenemos montado. Además confirma que la estructura diseñada compilar, sirve para realizar este tipo de test.

4.5. Círculo

Círculo es otro experimento básico que dibuja un círculo por pantalla partido en 256 partes iguales. También requiere de la librería *SDL_image*.

Para compilar este ejemplo realizamos los mismos pasos que con Cuadrado, pero debemos añadir a la variable *EE_LIBS* la librería matemática *-lm*, porque Círculo requiere de la función del coseno (*cos()*).

Este experimento también lo portamos correctamente y pinta por pantalla un círculo, gracias a esto podemos asegurar que las aplicaciones que solo pinten por pantalla podemos portarlas sin problemas.

4.6. Teclas

El objetivo es comprobar si efectivamente las SDL no responden a ningún estímulo del teclado como ha ocurrido cuando hemos realizado los test básicos del entorno de desarrollo.

Configuramos toda la estructura igual que el resto de experimentos y el resultado que obtenemos cuando ejecutamos Teclas es que la PS2 no reacciona ante ninguno de los eventos del teclado.

Después de este resultado llegamos a la conclusión de que las librerías del PS2DEV no tienen bien implementadas las funciones para capturar el teclado. Tendremos que tener en cuenta este error más adelante si es imprescindible utilizar el teclado para la portabilidad del videojuego.

4.7. Colisiones

El siguiente experimento es más complejo que el resto de experimentos realizados. Colisiones utiliza ficheros externos para cargar unas imágenes para el fondo de la pantalla y para la animación de las colisiones de objetos.

Este ejemplo cada vez que se pulsa la barra de espacio añade una esfera que comienza a colisionar con otras esferas que hayan en la pantalla, e incluso con los mismos bordes de la pantalla.

El procedimiento que seguimos para compilar Colisiones es el mismo que para el resto de experimentos. Pero esto no basta ya que Colisiones accede a ficheros externos, y estos tenemos que indicarle como encontrarlos. Como desconocemos como trata la PS2 los ficheros externos, decidimos realizar unos test de entrada/salida para entender el tratamiento de los ficheros externos.

En la sección [Tests de entrada/salida](#) explicamos todo lo referente a las pruebas que hacemos para probar como poder interactuar con ficheros externos.

Para que Colisiones funcione debemos añadir a las constantes que guardan el path de las imágenes el prefijo *host:*, un ejemplo de esto es:

```
DIR_ESFERA = "host:./imagenes/esfera.jpeg";  
DIR_FONDO = "host:./imagenes/fondo.jpeg";
```

Con el prefijo *host:*, la PS2 pide al PC que le facilite los ficheros que necesita vía remota.

Después de compilar este experimento sin problemas, podemos observar que al ejecutarlo devuelve un error cuando intenta cargar el segundo fichero. Esto es debido a que las librerías intentan cargar el segundo fichero en el mismo canal que tiene abierto para el primer fichero, este error bloquea la aplicación mostrando tan solo la primera imagen.

A la conclusión que llegamos después de obtener estos resultados es que el port de las SDL para la PS2 contiene errores en la implementación de la entrada/salida de ficheros.

4.8. Tests de entrada/salida

Los siguientes tests que realizamos se centran en probar la entrada por teclado, salida por pantalla y en el acceso a ficheros externos.

4.9. hello.c

Es un test que consiste en un código que recoge lo que escribimos por la entrada estándar del teclado y lo mostraba por la salida estándar de la pantalla del PC, no de la televisión.

```
/* hello.c */  
#include<stdio.h>  
int main()
```

```

{
    char c[20];
    int i;
    i = scanf("%s", c);
    printf("Hola %s scan %d \n", c, i);
    return 0;
}

```

Para compilar este ejemplo no necesitamos ninguna de las librerías SDL. El resultado que obtenemos es que la salida estándar muestra la palabra “Hola ” e ignora el teclado y la petición de “scanf”.

Este tercer intento de probar la entrada por teclado confirma que las librerías estándar de entrada y salida, en lo que se refiere a la entrada por teclado están mal implementadas.

4.10. helloFile.c

Con este test queremos probar a escribir en un fichero la cadena Hello y para ello mediante la función open abrimos un fichero para escribir en él.

```

/* Hello World program */
#include<stdio.h>
int main()
{
    FILE *fichero;
    fichero = fopen("ejemplo.txt", "w");
    fputs("Hello", fichero);
    fclose(fichero);
}

```

Para compilarlo utilizamos el mismo sistema que los demás test probado hasta el momento, y el resultado que obtenemos es que no sabe donde tiene que crear el archivo que necesita para escribir en él.

Al obtener este resultado decidimos realizar una segunda prueba que consiste en ejecutar este ejemplo desde el disco duro de la PS2, para descartar posibles errores del tratamiento de ficheros vía remota. Para poder lanzar este ejemplo desde el disco duro encontramos el Hdd Dump Tools un software que nos permite copiar y ejecutar aplicaciones desde el disco duro, pero el resultado que obtenemos es el mismo, no encuentra la ruta donde crear el archivo [\[URL.16\]](#).

Al obtener estos resultados llegamos a la conclusión de que en la PS2 se debe referenciar de manera especial los dispositivos para poder interactuar con ellos. El siguiente paso es investigar sobre este tema.

Lo primero que hacemos es buscar dentro de las fuentes del PS2DEV si algún fichero utiliza la función **fopen**. Y efectivamente encontramos que se utilizaba en el directorio de la librería **zlib** (librerías diseñadas para la compresión de ficheros).

En el fichero `zutil.h` vemos como están declaradas las distintas definiciones de `fopen` que debe de utilizar esta librería dependiendo del sistema operativo que utilice. Dentro de la carpeta de la `zlib` encontramos un directorio de test donde esta un fichero llamado `ejemplo.c` y en el que encontramos la siguiente definición:

```
#define TESTFILE "host:foo.gz"
```

Con esta línea el compilador indica a la PS2 que el fichero `foo.gz` se encuentra en el máquina (host) que esta ejecutando el `pksh`. A partir de este descubrimiento buscamos más información internet acerca de *host*: y encontramos que los demás dispositivos de PS2 también disponen de una forma para referenciarlos, en el caso del disco duro (`hdd`), las memory card (`mc0` y `mc1`), el `cdrom` y hasta el `usb` de la PS2 también tienen su propia forma de referenciarlos.

4.11. `helloFile.c`

(Con corrección de path)

Una vez encontramos como tenemos que referenciar a los ficheros externos para trabajar con ellos, decidimos modificar el ejemplo de *helloFile.c* adaptándolo a este nuevo cambio.

```
/* Hello World program */
#include<stdio.h>
int main()
{
    FILE *fichero;
    fichero = fopen("host:ejemplo.txt", "w");
    fputs("Hello", fichero);
    fclose(fichero);
}
```

Con esta modificación el resultado que obtenemos es un éxito, *helloFile* crea el archivo `ejemplo.txt` en el directorio de Linux donde se encuentra el ejecutable, y escribe dentro de él la cadena *“Hello”*.

Haber conseguido referenciar ficheros externos supone un gran avance para poder seguir con los experimentos de portabilidad.

4.12. `helloFileToFile.c`

Para hacer una última prueba respecto al tratamiento de ficheros externos que realiza la PS2, decidimos probar un programa al que le pasamos el nombre de dos ficheros y el programa copia todo lo del primer fichero al segundo fichero. Para ello abrimos el primer fichero como lectura y el segundo como escritura.

```

#include <stdio.h>
int main(int argc, char *argv[])
{
    FILE *fe, *fs;
    unsigned char buffer[2048]; //Buffer de 2 Kbytes
    int bytesLeidos;
    if(argc != 3) {
        printf ("Usar: copia <fichero_origen> <fichero_destino>");
        return 1;
    }
    fe = fopen(argv[1], "rb"); //Abrir el fichero de entrada
                                //en lectura y binario
    if(!fe) {
        printf ("El fichero %s no existe o no puede ser abierto.",\
                argv[1]);
        return 1;
    }
    fs = fopen(argv[2], "wb"); //Crear o sobrescribir el fichero
                                //de salida en binario
    if(!fs) {
        printf ("El fichero %s no puede ser creado.", argv[2]);
        fclose(fe);
        return 1;
    }
    //Bucle de copia:
    while((bytesLeidos = fread(buffer, 1, 2048, fe)))
        fwrite(buffer, 1, bytesLeidos, fs);
    //Cerrar ficheros:
    fclose(fe);
    fclose(fs);
    return 0;
}

```

Este experimento también funciono correctamente y con estos test podemos decir que ya sabemos como funciona la salida/entrada estándar ficheros en la PS2.

4.13. Test de portabilidad de un proyecto SDL que utiliza auto-tools

Finalmente explicamos todas las pruebas que hacemos para portar un videojuego desde Linux a PS2.

La idea principal es portar un videojuego que esté preparado para ser compilado con las auto-tools. En un principio el objetivo es probar a portarlo utilizando estas herramientas, de no poder ser así la idea es portarlo a partir de la creación de un Makefile específico para el videojuego que escogamos.

Juegos en SDL para Linux hay muchos en la propia web de las librerías de SDL podemos encontrar, como desconocemos hasta donde están portadas las SDL en PS2, decidimos buscar un juego sencillo que utilizara las librerías básicas de SDL y por el momento como mucho las `SDL_image` y las `SDL_mixer`, ya que hemos comprobado que funcionan.

También queremos que el juego que escojamos esté preparado para poder compilarlo con las autotools, ya que las autotools ofrecen la posibilidad de realizar compilación cruzada.

Tras mirar varios juegos que puedan servirnos para realizar el experimento inicial de portabilidad, decidimos que el juego que vamos a portar es Pongix [[PGX2006](#)], ver Figura 4.1.

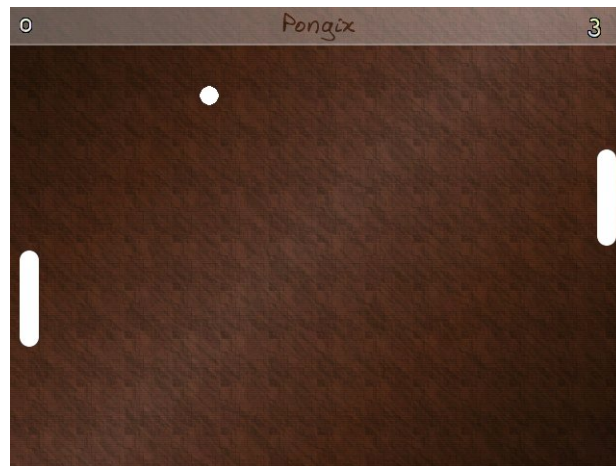


Figura 4.1: Pongix versión SDL de Pong para Linux

Pongix es una versión de Pong [[PON1972](#)] para Linux. Este juego que escogemos requiere de las `SDL_image` y las `SDL_mixer`, las cuales tenemos instaladas en el entorno después de haber ejecutado el test de Super Mario Wars (SMW).

Pongix lo encontramos en LosersJuegos [[URL.22](#)] página web dedicada a la programación en SDL. En esta página hay disponibles varios ejemplos y juegos sencillos programados en SDL para los distintos sistemas operativos del mercado.

De Pongix hay disponibles la versión 0.3 y la 0.4, decidimos que portaríamos era la versión 0.3, ya que la versión 0.4 utiliza la librería `SDL_net` porque incorpora la opción de juego multijugador por red. Al no conseguir encontrar información sobre la disponibilidad de esta librería en el port de PS2 decidimos optar por portar la versión inferior.

Una vez ya teníamos escogido qué juego queremos portar, el siguiente paso que hacemos es probar que las fuentes que hemos obtenido efectivamente compilan y ejecutan en Linux, para así descartar posibles errores de fuentes corruptas a la hora de realizar la portabilidad.

Al descomprimir las fuentes del Pongix encontramos los siguientes archivos:

```
aclocal.m4
```

```
config.guess
```

```
config.h.in
config.sub
configure
configure.ac
Makefile.am
Makefile.in
```

Como podemos observar entre estos archivos se encuentran todas las macros y los scripts necesarios para poder generar el archivo Makefile mediante el uso de las autotools.

La idea es, si podemos realizar la portabilidad con la ayuda de las autotools, entonces nos centraremos más en realizar el estudio sobre hasta donde están portadas las librerías de SDL de la PS2 y hasta donde podemos llegar con éstas.

Una vez comprobado que tenemos todos los archivos necesarios para poder generar el archivo Makefile con las autotools tan solo debemos ejecutar el siguiente comando:

```
>./configure
```

Al ejecutar configure las autotools, a partir de las macros y los scripts, prueba el sistema para comprobar que tenemos todo lo necesario para compilar Pongix y además configura el archivo Makefile de manera que este conozca todos los path necesarios donde encontrar los ficheros necesarios para poder realizar dicha compilación.

Tras obtener el archivo Makefile, compilamos Pongix y podemos comprobar que este funcionaba correctamente.

4.13.1. Autotools y compilación cruzada

Comprobadas las fuentes, el siguiente paso es estudiar que soporte o facilidades ofrece las autotools para realizar compilación cruzada. Comentamos en la sección dedicada a las [Autotools](#) en el capítulo de Análisis, que en la documentación de las autotools (en el autobook [\[URL.14\]](#)), existe un capítulo que hace referencia a la compilación cruzada.

El método que explican en este capítulo, que consiste en indicar a las autotools qué compilador es el escogido para compilar las herramientas propias de GNU/Linux y cual es el compilador que usamos para compilar todo lo que es propio de la plataforma a la que vamos a portar.

Aparte de los compiladores también es necesario indicarle donde se encuentran las librerías y los archivos de cabeceras del entorno de desarrollo, para poder indicarle a las autotools donde encontrar estos archivos, las autotools facilitan el uso de los flags LDLIBS y LDFLAGS.

Dicho esto la primera prueba que lanzamos para probar las autotools corresponde a la siguiente línea de comandos.

4.13. TEST DE PORTABILIDAD DE UN PROYECTO SDL QUE UTILIZA AUTOTOOLS39

```
./configure --build=$(./config.guess) --host=ee LDFLAGS="-mno-crt0 \  
-L$PS2SDK/ports/lib" LIBS="-lSDL_image -lSDL" 2>&1 | less
```

Con el parámetro `--host` le indicamos cual es el prefijo que tiene todas las herramientas que necesita el compilador y con los flags `LDFLAGS` y `LIBS` le indicamos que librerías debe incorporar al archivo Makefile que vamos a crear. Para poder ver mejor los resultados que nos devuelve la ejecución del comando `configure` decidimos redirigir con el comando `less` la salida estándar.

Según lo que leímos en el autobook [URL.14] referente a las autotools si tenemos bien declarados estos parámetros y si el entorno esta configurado correctamente, las autotools genera un archivo Makefile con el cual podemos compilar Pongix para la PS2.

Pero tras ejecutar esta primera prueba obtenemos el siguiente mensaje:

```
checking for a BSD-compatible install... /usr/bin/install -c  
checking whether build environment is sane... yes  
checking for gawk... no  
checking for mawk... mawk  
checking whether make sets $(MAKE)... yes  
checking for ee-strip... ee-strip  
checking for ee-gcc... ee-gcc  
checking for C compiler default output file name... a.out  
checking whether the C compiler works... yes  
checking whether we are cross compiling... yes  
checking for suffix of executables...  
checking for suffix of object files... o  
checking whether we are using the GNU C compiler... yes  
checking whether ee-gcc accepts -g... yes  
checking for ee-gcc option to accept ANSI C... none needed  
checking for style of include used by make... GNU  
checking dependency style of ee-gcc... gcc3  
checking for SDL_Init in -lSDL... no  
configure: error: * Can't find SDL library: http://www.libsdl.org
```

Como podemos observar en la salida del `less` el compilador y las diversas herramientas de nuestro entorno las encuentra sin problema alguno, pero sin embargo tiene un conflicto con la librería de SDL ya que el mensaje que devuelve dice que no la encuentra.

Este tipo de error principalmente podía ser debido a que `configure` no encuentra la librería. El mensaje que devuelve el `configure` es un mensaje genérico que se declaraba en una de las macros del autotools y este mensaje lo devuelve siempre que hay cualquier tipo de problema respecto a esas librerías.

Para averiguar la causa del problema que estaba devolviendo el `configure`, primero miramos que decía el archivo `config.log`, ya que en este archivo se encuentra todo el log de lo que ha realizado el comando `configure`.

En el fichero de log las líneas que interesan principalmente son las que contiene este fragmento:

```

configure:2827: ee-gcc -o conftest -g -O2 -mno-crt0
-T/usr/local/ps2dev/ps2sdk/ee/startup/linkfile -L/usr
/local/ps2dev/ps2sdk/ee/lib -L/usr/local/ps2dev/ps2sdk
/ee/startup/crt0.o conftest.c -lSDL >&5
/usr/local/ps2dev/ee/lib/gcc-lib/ee/3.2.2/../../../../
/ee/bin/ld: cannot find -lSDL
collect2: ld returned 1 exit status
configure:2833: $? = 1
configure: failed program was:
| /* confdefs.h. */
|
| #define PACKAGE_NAME "pongix"
| #define PACKAGE_TARNAME "pongix"
| #define PACKAGE_VERSION "0.3"
| #define PACKAGE_STRING "pongix 0.3"
| #define PACKAGE_BUGREPORT ""
| #define PACKAGE "pongix"
| #define VERSION "0.3"
| /* end confdefs.h. */
|
| /* Override any gcc2 internal prototype to avoid an error. */
| #ifdef __cplusplus
| extern "C"
| #endif
| /* We use char because int might match the return type of a gcc2
| builtin and then its argument prototype would still apply. */
| char SDL_Init ();
| int
| main ()
| {
| SDL_Init ();
| ;
| return 0;
| }
configure:2858: result: no
configure:2869: error: * Can't find SDL library: http://www.libsdl.org

```

Como podemos observar en estas líneas, configure no es capaz de compilar el test de la librería de SDL, porque no la encuentra.

Debido a este problema inicialmente decidimos ir añadiendo rutas al LDFLAGS, pero con esto solo complicamos más el comando y no corregimos el problema que tenemos.

Entonces lo que hacemos es extraer el código que el configure intenta ejecutar y lo copiamos en un archivo que decidimos llamarlo test1.c, el código que contiene este archivo es el siguiente:

4.13. TEST DE PORTABILIDAD DE UN PROYECTO SDL QUE UTILIZA AUTOTOOLS41

```
#include <stdio.h>
#include "SDL.h"
#ifdef __cplusplus
    extern "C"
#endif

char SDL_Init ();
int main ()
{
    SDL_Init ();
    ;
    return 0;
}
```

Ahora la idea es copiar el archivo test1.c en el directorio de los test básicos y modificamos el archivo Makefile para indicarle que también compile el archivo test1.c:

```
BINS = \
    testtimer.elf \
    testcdrom.elf \
    testendian.elf \
    testerror.elf \
    testjoystick.elf \
    testver.elf \
    checkkeys.elf\
    test1.elf
```

Al ejecutar make observamos el siguiente error:

```
ee-gcc -D_EE -O2 -G0 -Wall -I/usr/local/ps2dev/ps2sdk/ee/include \
-I/usr/local/ps2dev/ps2sdk/common/include -I. -I../include \
-I/usr/local/ps2dev/ps2sdk/ports/include/SDL -c test1.c -o test1.o
ee-gcc -mno-crt0 -T/usr/local/ps2dev/ps2sdk/ee/startup/linkfile \
-L/usr/local/ps2dev/ps2sdk/ee/lib -L/usr/local/ps2dev/ps2sdk/ports/lib \
-o test1.elf /usr/local/ps2dev/ps2sdk/ee/startup/crt0.o test1.o \
-L. -lc -L/usr/local/ps2dev/gskit/lib -L../lib -lsdl -lcdvd -lc -lsyscall \
-lkernel
**/usr/local/ps2dev/ps2sdk/ports/lib/libSDL.a(sdl_main.o)(.text+0x8):
In function 'main':main/ps2sdk/sdl_main.c: definiciones múltiples de 'main'
test1.o(.text+0x0):test1.c: primero se definió aquí
/usr/local/ps2dev/ps2sdk/ports/lib/libSDL.a(sdl_main.o)(.text+0x44):
In function 'main':main/ps2sdk/sdl_main.c: referencia a 'SDL_main'
sin definir**
collect2: ld devolvió el estado de salida 1
make: *** [test1.elf] Error 1
```

Como podíamos observar el test1 monta bien pero en el momento de compilar devuelve el mensaje de error de múltiple definición de main, este error se debe a que la librería SDL contiene

una función main y en el momento que nosotros intentamos compilar el test, el compilador encuentra un segundo main y por esta razón genera este error.

Como no entendemos para que han añadido un main a las librerías de SDL del port buscamos información por los foros de PS2 para ver si alguien comenta como solucionar este problema.

La solución es reescribir la definición del main pasándole los parámetros argc y argv para que las SDL.h del port no redefinen la función main. Esto supone un problema, ya que a los test del configure no podemos redefinirle la cabecera, porque el código de los test se generan de manera automática.

Tampoco investigamos mucho porque aunque consigamos que configure compile el test de las SDL, después configure devolverá un error en el momento que intente ejecutarlo ya que estamos desarrollando en una plataforma distinta a la que va destinado el test.

Las dos soluciones que encontramos son: intentar obtener un Makefile apto para compilar Pongix, portado a partir del Makefile generado por las autotools para Linux o quitar los test que configure realiza a la librería de SDL en el momento de generar el Makefile, para que así configure pueda seguir su proceso hasta el final.

La primera opción que decidimos probar es quitar los test de las librerías y ver que resultado obtenemos a partir de este experimento, para poder realizar esta prueba necesitamos tener instalados en Linux las libtool, el autoconf y el automake:

```
sudo su apt-get install libtool
sudo su apt-get install autoconf
sudo su apt-get install automake
```

Una vez tenemos instaladas estas aplicaciones lo primero que debemos hacer para que el configure no realice los test de las librerías es borrar del archivo configure.ac, las siguientes líneas:

```
AC_CHECK_LIB(SDL, SDL_Init, , \
    AC_MSG_ERROR([* Can't find SDL library: http://www.libsdl.org]))

AC_CHECK_LIB(SDL_image, IMG_Load, , \
    AC_MSG_ERROR([* Can't find SDL_image library: \
    http://www.libsdl.org]))
```

Estas líneas son las encargadas de comprobar si existen en el sistema las librerías que necesitaremos para poder compilar la aplicación, así que borrándolas el configure no testea ambas librerías.

Una vez modificamos el archivo y guardado, tan solo tenemos que ejecutar los comandos necesarios para volver a generar el configure con los nuevos cambios realizados, para regenerar el configure ejecutamos los siguiente comandos en el orden establecido, ya que cada comando va generando archivos que el siguiente comando necesita.

El primer comando que debemos ejecutar es aclocal este genera de nuevo el fichero aclocal.m4 que contiene algunas definiciones básicas para así después poder ejecutar el autoconf.

4.13. TEST DE PORTABILIDAD DE UN PROYECTO SDL QUE UTILIZA AUTOTOOLS43

```
aclocal
```

Después tenemos que ejecutar el automake el cual genera los archivos que necesita tanto el autoconf como el mismo automake, para ejecutar este comando usamos la siguiente instrucción:

```
automake --add-missing
```

Y por último tan solo falta ejecutar el autoconf para que vuelva a generar el archivo configure con las nuevas opciones añadidas:

```
autoconf
```

Siguiendo estos pasos ya tenemos en el directorio de Pongix el configure nuevo que generamos, este no realiza los test que impiden que podamos generar el Makefile.

Así que el siguiente paso que decidimos seguir es ejecutar la línea de comando del configure que hemos ejecutado antes y el resultado que obtuvimos ahora es un archivo Makefile generado con el nuevo configure, pero este Makefile no garantiza que Pongix compile sin problema alguno.

```
./configure --build=$(./config.guess) --host=ee LDFLAGS="-mno-crt0 \  
-L$PS2SDK/ports/lib" LIBS="-lSDL_image -lSDL" 2>&1 | less
```

Ejecutamos el comando make y como resultado obtenemos el siguiente error:

```
cd . && autoheader  
cd . \  
    && CONFIG_FILES= CONFIG_HEADERS=config.h \  
    /bin/bash ./config.status  
config.status: creating config.h  
config.status: executing default-1 commands  
make all-recursive  
make[1]: se ingresa al directorio '/home/rul/PFC/pruebas/sdl_ps2  
/pongix-0.3'  
Making all in src  
make[2]: se ingresa al directorio '/home/rul/PFC/pruebas/sdl_ps2  
/pongix-0.3/src'  
ee-gcc -DHAVE_CONFIG_H -I. -I. -I.. -DDATADIR=\""/usr/local/share  
/pongix/"\" -g -O2 -c pongix.c  
En el fichero incluido de mundo.h:22,  
    de pongix.c:19:  
**util.h:22:21: SDL/SDL.h: No existe el fichero ó directorio**  
**util.h:23:27: SDL/SDL_image.h: No existe el fichero ó directorio**  
En el fichero incluido de mundo.h:22,  
    de pongix.c:19:  
util.h:29: error de decodificación antes del elemento '*'  
util.h:29: aviso: la definición de datos no tiene tipo o clase de  
almacenamiento
```

```
util.h:31: error de decodificación antes del elemento '*'
util.h:31: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
util.h:32: error de decodificación antes del elemento '*'
En el fichero incluido de mundo.h:23,
de pongix.c:19:
dirty.h:27: error de decodificación antes de "SDL_Rect"
dirty.h:27: aviso: no hay punto y coma al final del struct o union
dirty.h:29: error de decodificación antes del elemento '}'
dirty.h:29: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
dirty.h:32: error de decodificación antes del elemento '*'
dirty.h:32: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
dirty.h:33: error de decodificación antes del elemento '*'
dirty.h:34: error de decodificación antes del elemento '*'
dirty.h:35: error de decodificación antes del elemento '*'
dirty.h:36: error de decodificación antes del elemento '*'
dirty.h:37: error de decodificación antes del elemento '*'
En el fichero incluido de mundo.h:24,
de pongix.c:19:
fuente.h:30: error de decodificación antes de "SDL_Rect"
fuente.h:30: aviso: no hay punto y coma al final del struct o union
fuente.h:32: error de decodificación antes del elemento '*'
fuente.h:32: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
fuente.h:34: error de decodificación antes del elemento '}'
fuente.h:34: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
fuente.h:37: error de decodificación antes del elemento '*'
fuente.h:37: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
fuente.h:38: error de decodificación antes del elemento '*'
fuente.h:39: error de decodificación antes del elemento '*'
fuente.h:40: error de decodificación antes del elemento '*'
fuente.h:41: error de decodificación antes del elemento '*'
fuente.h:42: error de decodificación antes del elemento '*'
fuente.h:43: error de decodificación antes del elemento '*'
En el fichero incluido de pongix.c:19:
mundo.h:30: error de decodificación antes de "SDL_Surface"
mundo.h:30: aviso: no hay punto y coma al final del struct o union
mundo.h:31: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
mundo.h:34: error de decodificación antes del elemento '*'
mundo.h:34: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
mundo.h:39: error de decodificación antes del elemento '}'
```

4.13. TEST DE PORTABILIDAD DE UN PROYECTO SDL QUE UTILIZA AUTOTOOLS45

```

mundo.h:39: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
mundo.h:42: error de decodificación antes del elemento '*'
mundo.h:42: aviso: la definición de datos no tiene tipo o clase de
almacenamiento
mundo.h:43: error de decodificación antes del elemento '*'
mundo.h:44: error de decodificación antes del elemento '*'
mundo.h:45: error de decodificación antes del elemento '*'
mundo.h:46: error de decodificación antes del elemento '*'
mundo.h:47: error de decodificación antes del elemento '*'
mundo.h:48: error de decodificación antes del elemento '*'
mundo.h:49: error de decodificación antes del elemento '*'
pongix.c: En la función 'main':
pongix.c:25: operadores inválidos para el binario *
pongix.c:29: aviso: asignación de tipo de apuntador incompatible
pongix.c:34: apuntador deferenciado a tipo de dato incompleto
make[2]: *** [pongix.o] Error 1
make[2]: se sale del directorio '/home/rul/PFC/pruebas/sdl_ps2
/pongix-0.3/src'
make[1]: *** [all-recursive] Error 1
make[1]: se sale del directorio '/home/rul/PFC/pruebas/sdl_ps2
/pongix-0.3'
make: *** [all-recursive-am] Error 2

```

Este error lo que esta diciendo es que no puede encontrar los archivos de cabecera SDL que le hemos dicho que utilice, para ser exactos los esta buscando en un directorio SDL dentro de la raíz de Pongix, para poder solventar este problema temporalmente decidimos crear un enlace simbólico a las SDL del PS2SDK con el siguiente comando para así poder ver si compila Pongix.

```
>ln -s /usr/local/ps2dev/ps2sdk/ports/include/ SDL
```

Ahora *make* ya encuentra los archivos de cabecera, pero aún así devuelve el siguiente error:

```

make all-recursive
make[1]: se ingresa al directorio '/home/rul/PFC/Ejemplos/pongix'
Making all in src
make[2]: se ingresa al directorio '/home/rul/PFC/Ejemplos/pongix/src'
ee-gcc -g -O2 -mno-crt0 -L/usr/local/ps2dev/ps2sdk/ports/lib -o pongix
pongix.o util.o pelota.o dirty.o paleta.o mundo.o menu.o juego.o fuente.o
cursor.o -lSDL_image -lsdl
/usr/local/ps2dev/ee/bin/./lib/gcc-lib/ee/3.2.2/./././././ee/bin/ld:
aviso: no se puede encontrar el símbolo de entrada _start; usando por
defecto 000000000100040
util.o(.text+0x74): In function 'iniciar_sdl':
/home/rul/PFC/Ejemplos/pongix/src/util.c:33: undefined reference to
'printf' util.o(.text+0x164): In function 'cargar_imagen':
/home/rul/PFC/Ejemplos/pongix/src/util.c:71: undefined reference to

```

```
'memset' util.o(.text+0x170):/home/rul/PFC/Ejemplos/pongix/src/util.c:73:
undefined reference to 'strcat'
util.o(.text+0x1f0):/home/rul/PFC/Ejemplos/pongix/src/util.c:79:
undefined reference to 'printf'
pelota.o(.text+0x1c): In function 'dist':
/home/rul/PFC/Ejemplos/pongix/src/pelota.c:25: undefined reference to
'sqrt' pelota.o(.text+0x4c): In function 'pelota_iniciar':
/home/rul/PFC/Ejemplos/pongix/src/pelota.c:31: undefined reference to
'malloc'
...
```

Como podemos observar de las cosas que se queja es que le falta referencias a librerías, lo cual indica que el Makefile que generamos no esta de forma correcta, ya que falta por cubrir dependencias.

El siguiente paso que realizamos para intentar afrontar este nuevo inconveniente, es ir añadiendo a los FLAGS de configure las librerías que vayamos necesitando según va pidiendo Pongix, y ir generando de nuevo el Makefile con esos FLAGS nuevos.

Después de varias pruebas con distintas combinaciones de flags, encontramos los flags adecuados para resolver todas las dependencias de Pongix:

```
>./configure --build=$(./config.guess) --host=ee LDFLAGS="-mno-crt0 \
-T/usr/local/ps2dev/ps2sdk/ee/startup/linkfile \
/usr/local/ps2dev/ps2sdk/ee/startup/crt0.o" LIBS="-L. \
-L/usr/local/ps2dev/ps2sdk/ee/lib -L/usr/local/ps2dev/ps2sdk/ee/lib \
-L/usr/local/ps2dev/gskit/lib -L../lib \
-L/usr/local/ps2dev/ps2sdk/ports/lib -L/usr/local/ps2dev/ee/lib \
-lSDL_image -lsdl -lsdlmain -ljpeg -lpng -lz -ltiff -lmc -lm -lc \
-lkernel" 2>&1 | less
```

Al ejecutar el comando make con el nuevo fichero Makefile que obtenemos a partir de esta línea de comando, conseguimos como resultado un fichero Pongix dentro del directorio src. Ahora ya tenemos un ejecutable compilado en Linux con las herramientas de compilación de la PS2 que es lo que pretendemos conseguir.

Ahora tan solo tenemos que probar en la PS2 este Pongix, para ver si el ejecutable que acabamos de obtener funcionaba correctamente. El siguiente paso es lanzar el loader pksh para conectar con la PS2 y ejecutar el Pongix que acabamos de compilar.

Lo primero que debemos ejecutar es el comando pksh y de esta manera entramos en la consola de pksh:

```
>pksh
```

Una vez estamos dentro lo siguiente que hacemos es lanzar Pongix:

```
pksh>cd src (el ejecutable que configure nos ha creado se encuentra
dentro del directorio src de la carpeta pongix)
pksh>execee pongix
```

4.13. TEST DE PORTABILIDAD DE UN PROYECTO SDL QUE UTILIZA AUTOTOOLS47

En la pantalla de PS2 no obtenemos ningún resultado y en la salida de nuestro monitor obtenemos el siguiente mensaje :

```
pksh> execee pongix
log: loadelf: fname host:pongix secname all
log: loadelf version 3.30
log: Input ELF format filename = host:pongix
log: 0 00100000 000c20ac .....
log: Loaded, host:pongix
log: start address 0x1000e0
log: gp address 00000000
log: SDL: initializing gsKit in PAL mode
log: SDL_SetVideoMode 640 x 480 x 16
log: SDL_Video: local texture allocated at 0x00424180
log: SDL_video: centered surface of (640, 480) onto a
screen of (640, 512) at (0, 16)
log: SDL_video: ratio of 1:1.000, rastered surface is (640, 480)
log: SDL: HW Surface Flipping !
log: open name host:imagenes/fondo_juego.jpg flag 1 data 45a78
log: open fd = 2
log: open name host:imagenes/pelota.bmp flag 1 data 45a78
log: open fd = 2
log: open name host:imagenes/pelota.bmp flag 1 data 45a78
log: open fd = 2
log: open name host:imagenes/paleta.bmp flag 1 data 45a78
log: open fd = 2
log: open name host:imagenes/paleta.bmp flag 1 data 45a78
log: open fd = 2
log: open name host:imagenes/fuente.bmp flag 1 data 45a78
log: open fd = 2
log: Fuente: se encontraron 112 letras
log: + cargando: host:imagenes/fuente.bmp
log: Reiniciando el men
pksh>
```

Llegados a este punto volvemos a encontrar con un nuevo obstáculo para conseguir la portabilidad de Pongix.

Decidimos analizar el código de Pongix y vemos que utiliza unos archivos de imagen para el fondo de pantalla y el resto de dibujos que aparecen. Esto supone un inconveniente porque como hemos comprobado en los test anteriores de portabilidad las SDL no son capaces de cargar más de un archivo de imagen.

Y después de este experimento llegamos a un par de conclusiones:

- Es difícil usar autotools pensando en compilación cruzada. Sobretodo porque hay test que las autotool necesita ejecutar en la máquina de destino y no en la de desarrollo
- El port de SDL para PS2 contiene errores en la implementación de la entrada/salida.

Por lo tanto la tarea de portar Pongix se hace muy complicada y requiere de tiempo adicional para poder llegar a portarse.

Capítulo 5

Valoración económica

5.1. Costes de hardware y software

La inversión económica de este proyecto a lo que se refiere al material que se necesita para llevar a cabo los experimentos como podemos observar en las siguientes listas es prácticamente la misma, al margen de que se desarrolle el proyecto con una PS2 o una PSTwo (PS2 Slim):

Componente	Precio
Videoconsola PS2	149.99
Tarjeta de memoria	19.95
Tarjeta de ethernet	39.95
Disco duro 120Gb	40.50
Flash memory 1Gb	9.90
Cable ethernet cruzado o paralelo	5.00
Modchip (DMS4)	100.0
Bobina 10 CD aprox.	4.50
Total	369.79

Componente	Precio
Videoconsola PSTwo	129.95
Tarjeta de memoria	19.95
Disco duro 200Gb Externo USB	89.69
Flash memory 1Gb	9.90
Cable ethernet cruzado o paralelo	5.00
Modchip (DMS4 Lite PRO) <i>dependiendo del sistema de protección de subida de tensión</i>	100.0
Bobina 10 CD aprox.	4.50
Total	369.79

Como podemos observar ambas configuraciones tienen un coste similar, tan solo puede variar dependiendo del modchip que instalemos.

Para llevar a cabo este proyecto utilizamos la primera configuración, porque disponemos de la mayor parte de los componentes necesarios.

5.2. Costes de dedicación

Aparte del coste de los materiales también tenemos que tener en cuenta el coste invertido en horas de trabajo dedicadas.

Todas las tareas que realizamos durante toda el desarrollo del proyecto las podemos englobar en tres tareas principales las cuales detallamos a continuación:

Búsqueda de información

Esta tarea es la que mayor peso en horas de dedicación realizamos, ya que durante todo el proyecto nos encontramos ante la necesidad constante de tener que buscar información, ya sea para ver como instalar uno de los diversos entornos testeados o hasta para ver las posibles soluciones a los bugs que vamos encontrando.

Además también tenemos que buscar toda la información que hay disponible sobre la Compilación Cruzada e incluso para poder realizar esta memoria también tenemos que buscar la documentación del ReStructuredText [RST2006] (lenguaje de marcas evolución del Structured Text) ya que era un requisito no funcional del proyecto.

Experimentos

Esta tarea de igual modo que la búsqueda de información ocupado la mayor parte de la dedicación del PFC.

Todo este tiempo dedicado, es debido a que desconocemos todos los entornos o aplicaciones que utilizamos. Además en algunos al no haber suficiente documentación, nos vemos obligado a probar con mucho detenimiento todo lo que utilizábamos para asegurarnos y comprender el funcionamiento.

Las primeras pruebas que realizamos son en la fase inicial de búsqueda del entorno de desarrollo. Una vez escogemos el entorno de desarrollo también testeamos todas las librerías y herramientas que necesitamos para montar este entorno.

Los experimentos más importantes en el desarrollo del proyecto son los realizamos durante el proceso de portabilidad de Pongix, este proceso es largo debido a las librerías del port de SDL de PS2, que no están bien portadas, y esto nos causa problemas constantes.

Aparte de todos estos experimentos también decidimos como posible tarea futura, que tras portar Pongix a PS2, estudiaríamos portar aplicaciones cada vez de mayor complejidad para ver que nos podía llegar a ofrecer el port de SDL.

Documentación

Por último no podemos olvidarnos de la documentación de cada prueba realizada o de cada proceso de instalación de cada entorno de desarrollo. Por la naturaleza de este proyecto durante toda la realización de él, tenemos que ir constantemente documentando todo lo que llevábamos a cabo para después sintetizarlo todo en la memoria del proyecto.

5.3. Detalles de las tareas realizadas

A continuación en la siguiente tabla podemos encontrar un listado con la relación de todas las tareas realizadas, al lado de cada tarea indicamos las horas dedicadas. Para poder calcular este número de horas invertidas hacemos una estimación de número de horas por semana y aproximadamente son una media de 22 horas semanales.

Tarea realizada	Tiempo
Búsqueda de información (Definición del PFC)	44h
Búsqueda de información (Entornos de desarrollo para PS2)	132h
Instalación de los entornos de desarrollo	176h
Búsqueda de información sobre Compilación Cruzada	88h
Redactar el informe preliminar del proyecto	22h
Implantación, testeo y documentación del entorno de desarrollo escogido	110h
Estudio de la Compilación cruzada a partir de ejemplos	33h
Búsqueda de la aplicación a portar	33h
Estudio de las autotools	44h
Busqueda de aplicaciones simples para pruebas	44h
Pruebas de portar la aplicación escogida	220h
Desarrollo de la memoria a partir de las notas y de los resultados obtenidos	132h
Preparar la presentación del PFC	44h
<i>Horas totales invertidas</i>	<i>1122h</i>

Tras una visión global de las tareas que componen este proyecto en el siguiente apartado explicamos con un poco más de detalles cada una de las tareas realizadas.

La primera tarea que realizamos consistió en buscar información para decidir la definición del proyecto, para esta tarea decidimos dedicar un mínimo de dos semanas porque de ella dependería el desarrollo del proyecto.

Una vez decidida la PS2 como plataforma de trabajo dedicamos aproximadamente seis semanas a buscar los diversos entornos de desarrollo que existen en PS2 y documentarnos de como se deben instalar. Tras estudiar las ventajas e inconvenientes que ofrecen estos entornos de desarrollo, decidimos utilizar el PS2DEV con su port de las librerías de SDL para estudiar la portabilidad de una aplicación de Linux a PS2.

Una vez organizada toda la información obtenida y filtrada, tan solo la que es útil, instalamos cada uno de los entornos de desarrollo y anotamos en un borrador los pasos que vamos realizando. Además también anotamos las ventajas e inconvenientes que aportan cada uno de los entornos a la hora de desarrollar aplicaciones en él. Esta larga tarea nos lleva ocho semanas.

A partir de este momento comienza el estudio de la portabilidad de Pongix. Estamos dos semanas buscando por internet métodos para realizar portabilidad de aplicaciones entre distintas plataformas, y el método que escogemos es la Compilación cruzada. Para entender y conseguir todo lo necesario para la compilación cruzada dedicamos dos semanas más.

Llegados a este punto del proyecto el siguiente paso que realizamos es redactar el informe preliminar del proyecto, al que le dedicamos una semana.

Ya decidido que el entorno que usaremos es el PS2DEV, decidimos realizar toda la instalación de este entorno en una máquina con una Ubuntu recién instalada, y redactar todos los pasos necesarios para instalar este entorno y cada una las librerías que usamos en él. También realizamos todas las pruebas necesarias para asegurar el correcto funcionamiento del entorno, en esta tarea invertimos cinco semanas.

Después de haber realizado todas estas tareas el 80 % de la fase de buscar información ya estaba completada y el 60 % de la fase de los experimentos también. Entonces el siguiente paso es realizar la portabilidad de Pongix. Este proceso consiste en un proceso cíclico de experimentos, búsqueda de información y de documentación de cada prueba. Esta tarea nos lleva diez semanas, aunque podríamos haber dedicado más tiempo de no ser por las limitaciones que nos encontramos en las librerías de SDL para PS2.

Por último tardamos seis semanas en dejar lista la memoria del proyecto y un dos de semanas más la presentación.

Capítulo 6

Conclusiones

Haciendo un balance de los objetivos que propusimos si recapitulamos, la principal meta era realizar el estudio de la **Portabilidad de un videojuego de Linux a PS2**.

Para conseguir alcanzar este objetivo primero marcamos otras tareas intermedias igual de importantes o más que el objetivo principal. De estas tareas la primera de ellas fue el estudio y puesta en marcha de los diversos entornos de desarrollo disponibles para PS2. En el siguiente objetivo englobamos realizar todos los test necesarios para probar tanto el entorno de programación como el lenguaje escogido y a las librerías que en este caso práctico fueron las SDL. Y el último de estos objetivos secundarios que nos marcamos fue el crear la documentación de todas las tareas realizadas sintetizando toda esta información en un solo documento redactado con **reStructuredText**.

De todos los objetivos que nos propusimos, podemos decir que el primero de ellos lo completamos con éxito tras un largo proceso de recolección de información, elección de toda la información que pudiera resultar nos útil y después la puesta en práctica de esta información para configurar cada uno de los entornos de desarrollos libres que encontramos disponibles. Con la información obtenida instalamos cada uno de ellos y estudiamos sus ventajas e inconvenientes y escogimos el que nos resulto más adecuado para nuestro propósito, en este caso práctico elegimos como método de desarrollo la **Programación RAW** y como entorno de desarrollo el **PS2DEV** el cual consistía en un conjunto de aplicaciones y librerías implementadas bajo licencias de libre distribución, tal y como requería la especificación del proyecto.

La siguiente tarea consistía en poner en marcha el PS2DEV y testear con ejemplos sencillos su correcta configuración. Podemos asegurar que este objetivo lo conseguimos superar.

Para comprobar el funcionamiento del port de SDL que instalamos a partir del PS2DEV, ejecutamos una serie de test que las fuentes del port llevaban incorporadas en un directorio llamado **test**, a parte de estos test pudimos observar en un principio que la entrada por teclado y ratón no los controlaba de manera correcta pero por lo menos la entrada del DualShock sí que eramos capaces de interactuar con ella.

También obtuvimos las fuentes de un videojuego programado en SDL y disponible para PS2, el Super Mario Wars videojuego implementado con SDL, y que nos ayudo a testear el entorno y las librerías **SDL_image** y **SDL_mixer** dos de las más utilizadas para estos tipos de desarrollos.

Respecto a realizar toda la documentación de los pasos que llevamos a cabo y redactarla

utilizando reStructuredText ese objetivo lo completamos.

El **Estudio de la portabilidad de un videojuego de Linux a PS2**, tenemos que decir que Pongix el videojuego que escogimos para portar utilizando el port de SDL del PS2DEV no conseguimos finalmente portarlo y mucho menos utilizando las autotools, por dos motivos:

1. Es difícil usar autotools pensando en la Compilación cruzada. Hay tests de las autotools que deberían ejecutarse en la máquina de destino y no en la de desarrollo.
2. El port de SDL para PS2 contiene errores en la implementación de la entrada y salida.

Por otro lado sería importante destacar que durante el estudio de la portabilidad fuimos capaces de portar las aplicaciones **cuadrado** y **circulo**, por eso podemos decir que el entorno era capaz de portar aplicaciones entre GNU/Linux y PS2, pero tras numerosos contratiempos vimos que tenían que ser aplicaciones simples que no utilizaran ficheros externos.

Por falta de tiempo para poder invertir más en la investigación de como portar Pongix, podríamos decir que a partir de esta documentación se podría continuar la investigación en un futuro. Para no perder todo el tiempo que ya le dedicamos nosotros, la futuras tareas que podrían ayudar a la portabilidad serían en un principio dos:

1. Investigar el error que devuelve las librerías a la hora de intentar cargar más de dos ficheros de imágenes, ya que disponemos en el repositorio de las fuentes de estas librerías.
2. Usar la librería que implemento el creador de Super Mario Wars, y estudiar cual es el proceso que sigue para gestionar el problema de la carga de ficheros de imágenes.

Bibliografía

- [FSF1996] The free software foundation. The free software definition. 1996. <http://www.gnu.org/philosophy/free-sw.html>
- [PS22000] PlayStation 2. Sony PlayStation 2. 2000. <http://en.wikipedia.org/wiki/Playstation2>
- [SON1945] Sony. Sony. 1945. <http://en.wikipedia.org/wiki/Sony>
- [GNU1996] The free software foundation. El sistema operativo GNU. 1996. <http://www.gnu.org/home.es.html>
- [LIN2002] Linux. Linux. 2002. <http://en.wikipedia.org/wiki/Linux>
- [PS2SDK] PS2SDK. <http://ps2dev.org/PS2/Projects/ps2sdk>
- [UBU2004] Ubuntu. Ubuntu-es. 2006. <http://www.ubuntu-es.org>
- [SDL2006] Simple DirectMedia Layer (SDL). 2006 <http://www.libsdl.org/index.php>
- [SMW2006] Playstation 2 version by Evil0 and the Froggies. Super Mario Wars. 2006. <http://psxdev.info/evilo/smw.html>
- [PGX2006] LosersJuegos (GPL). Pongix (Versión de Pong para Linux). 2006. <http://www.losersjuegos.com.ar/juegos/pongix/pongix.php>
- [DSK2004] DualShock. DualShock. 2004. <http://en.wikipedia.org/wiki/DualShock>
- [EYE2004] EyeToy. EyeToy. 2004. <http://en.wikipedia.org/wiki/EyeToy>
- [SIN2004] SingStar. SingStar. 2004. <http://en.wikipedia.org/wiki/SingStar>
- [GUI2005] Guitar Hero. 2005. http://en.wikipedia.org/wiki/Guitar_Hero_%28video_game%29
- [DOC.01] ps2dev.ofcode.com by Neofar. Arquitectura y Programación PS2. http://ps2dev.ofcode.com/modules/wordpress/?page_id=37
- [URL.01] DMS Technologies. <http://www.dms3.com/>
- [URL.02] ToxicOs. (Enlace perdido). <http://www.teamtoxic.com.cn/>
- [URL.03] Swap Magic 3. <http://www.swapmagic3.com/>

- [URL.04] Manual de Cogswap y carga de backups con Swap Magic. <http://www.planetadejuego.com/foros/manual-de-cogswap-y-carga-de-backups-con-swap-magic-1418.html>
- [URL.05] HD Advance. <http://www.hdadvance.com/>
- [URL.06] HDD Dump Tool. <http://www.dms3.com/dms4se/downloads.html>
- [URL.07] PCSX2. <http://www.pcsx2.net/>
- [URL.08] Guía de configuración del PCSX2. http://www.pcsx2.net/guide.php?lang=es_ES
- [URL.09] Volcador de BIOS de PS2. BIOS Dumper. <http://www.pcsx2.net/downloads.php?p=tool>
- [URL.10] PS2DEV. Loaders PS2. <http://ps2dev.org/>
- [DEB1997] Debian. Debian GNU/Linux. 1997. <http://www.debian.org/index.es.html>
- [URL.11] Tutorial de instalación de una distribución Linux en PS2 (BlackRhino). <http://www.xtremett.com/modules.php?name=Content&pa=showpage&pid=5>
- [URL.12] BlackRhino. <http://blackrhino.xrhino.com>
- [URL.13] Repositorio svn de PS2DEV. <http://svn.ps2dev.org>
- [AUT2003] Autotools. GNU build system. 2003. http://en.wikipedia.org/wiki/GNU_build_system
- [URL.14] GNU Autoconf, Automake and Libtool. Cross Compilation with GNU Autotools. http://sourceware.org/autobook/autobook/autobook_258.html
- [URL.15] SDL. Simple DirectMedia Layer. <http://www.libsdl.org/>
- [URL.16] GCC. The GNU Compiler Collection. <http://gcc.gnu.org/>
- [URL.17] Pukklink shell (pksh). PS2DEV. http://ps2dev.org/ps2/Loaders/PC_side_clients
- [URL.18] Ps2link v1.46. http://ps2dev.org/PS2/Loaders/PS2_side_boot_loaders/
- [URL.19] ElOtroLado.net. <http://www.elotrolado.net/>
- [URL.20] Mini tutorial: Crear un boot cd by Suloku. ElOtroLado.net. http://www.elotrolado.net/hilo_mini-tutorial-crear-un-boot-cd_462421
- [URL.21] Toolchain. GNU Toolchain. http://en.wikipedia.org/wiki/GNU_toolchain
- [RST2006] RestructuredText. Markup Syntax and Parser Component of Docutils. <http://docutils.sourceforge.net/rst.html>
- [PON1972] Pong. Atari. 1972. <http://en.wikipedia.org/wiki/Pong>
- [URL.22] LosersJuegos. <http://www.losersjuegos.com.ar/principal/principal.php>
- [NIN1889] Nintendo. Nintendo. 1989. <http://en.wikipedia.org/wiki/Nintendo>