BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"IMPLEMENTACIÓN DEL SISTEMA PC-BSD EN EL ENTORNO OPERACIONAL DE OPENBSD"

TESIS PROFESIONAL

PARA OBTENER EL TÍTULO DE

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
FREDI VICTORIA JUÁREZ

ASESOR:
MICHO DURDEVICH

COASESOR:
HILDA CASTILLO ZACATELCO

PUEBLA, PUEBLA. 2008

Agradecimientos

Todo lo que vale la pena lleva su tiempo, tiempo de preparación, tiempo de información, tiempo de elaboración. Es por ello que dedico el presente trabajo a mi familia, amigos y demás personas que siempre creyeron en mi. Personas importantes en esos momentos difíciles por mi paso en la universidad.

Quiero agradecer a mis maestros: Hilda Castillo Zacatelco por demostrarme el alcance y posibilidades de la programación. A Maria Esther Arroyo Castelazo por el apoyo en mis primeros años de carrera pero sobre todo a los consejos de mi asesor y amigo Micho Durdevich.

Por último, quiero mencionar a dos instituciones que fueron importantes en el desarrollo de mi trabajo de tesis. Doy gracias al ***Instituto de Matemáticas de la UNAM***, quienes me apoyaron en múltiples formas, en particular proporcionándome acceso a sus instalaciones en donde se desarrolló gran parte del proyecto, y agradesco al ***Consejo de Ciencia y Tecnología de Estado de Puebla (CONCYTEP)***, por su apoyo y por todo el tiempo otorgado durante la realización del proyecto, a través de la **Beca-Tesis Concytep 2007**.

Resumen

El presente trabajo muestra soluciones e ideas a futuro de como puede transformarse al sistema operativo **OpenBSD** (el sistema operativo más seguro del mundo) para dejar de considerarse un sistema operativo poco amigable y darlo a conocer como una alternativa para su uso con usuarios finales.

Se tomará como base del proyecto el caso de **PC-BSD** (distribución basada en **FreeBSD**) para generar un concepto similar, el cuál nos llevará a introducir una nueva distribución llamada **X-BSD**. Dentro del nuevo sistema, se incluyen ciertas características del sistema operativo **Mac OS X** para ofrecer más mejoras y mayor de seguridad que **FreeBSD** no tiene y **OpenBSD** puede aportar.

Palabras clave: Sistema Operativo, kernel, compilación, script, puerto, paquete, librería, dispositivo, aplicación.

Índice general

Capítulo 1

Introducción

La historia de los sistemas **BSD** comienza con el trabajo de empleados de **AT&T**. Empleados quienes crean al sistema operativo **Unix** en los comienzos de 1970. Durante éste tiempo, las compañias telefónicas fueron olvidando la competencia en la industria de las computadoras. Las compañías de telecomunicaciones utilizaban **Unix** internamente. A pesar de ello, no podían transformar a éste en un producto comercial. Tal fue así, que **AT&T** empezó a conceder licencias a las instituciones académicas de manera gratuita o por un pago simbólico. Tal acción resulto conveniente para todas las partes: **AT&T** obtuvo un poco dinero, una generación de científicos en computadoras apostaban sus desarrollos sobre la tecnología de **AT&T**, las universidades evitaron los altos costos de licencia del sistema operativo, y los estudiantes fueron capaces de indagar sobre el interior del código fuente y ver como trabajaban realmente las computadoras con **Unix**.

Comparando al original **Unix** con algunos sistemas operativos de esa época, éste no fue muy bueno. Pero todos aquellos estudiantes que tenían acceso al código fuente, podían implementar partes que ellos necesitaban. Si un maestro encontraba un error, el podía asistir a sus estudiantes para reparar al sistema. Si alguien necesitaba alguna nueva característica, uno podría construirla debido a tener acceso al código fuente del sistema. Con el crecimiento de **Internet** en los comienzos de los 1980, éstas nuevas características fueron intercambiándose entre universidades en forma de parches. La ***Computer Science Research Group (CSRG)*** de la Universidad Berkeley de California actuó como una unidad central para éstos parches distribuyéndolos a cualquiera que los quisiera con una licencia valida del código fuente de

AT&T. El resultado de ésta colección de parches llego a ser conocido con el nombre de ***Berkeley Software Distribution*** o simplemente **BSD**.

Tras quince años de continuo desarrollo por estudiantes de los mejores programas en ciencias de la computación, hicieron que el sistema fuera madurando completamente. Hecho que dío como resultado que al menos todos quienes usaban **Unix** estaban realmente usando **BSD**. La **CSRG** estuvo totalmente sorprendida, puesto que al final de éstos años, se encontró con la sorpresa de haber reemplazado al menos todo el código original de **AT&T**.

Por otra parte, como la **CSRG** estuvo trabajando en la mejora de los productos de **AT&T**. La compañía de telecomunicaciones estuvo haciendo su propio trabajo de desarrollo de **Unix** al conocer sus necesidades internas. Así, los desarrolladores de **AT&T** implementaron tales características y al mismo tiempo también evaluaban los parches que venían desde el **CSRG**. Cuando ellos vieron al proyecto en una fase ya madura, **AT&T** tomó gran parte del código **BSD** y lo incorporó dentro del **Unix** de **AT&T**. Hecho lo anterior, relicenciaron éste resultado para las universidades, quienes usaban al sistema como base para su siguiente ronda de trabajo.

Mientras tanto, cuando la **CSRG** iba a llegar a su fin, varios de sus miembros crearon una empresa llamada ***Berkeley Software Design Incorporated (BSDI)***. Empresa que desarrollaría una versión del código de **CSRG** con soporte comercial. Cuando **BSDI** comercializó y ofreció sus servicios de soporte para su versión del sistema llamada **BSD/386**, a un precio muy inferior al de **AT&T**. ***Unix System Laboratories*** (la filial de **AT&T** creada para desarrollar y comercializar **Unix**) interpuso una demanda contra **BSDI** y la Universidad de California por violación de copyright y divulgación de secretos comerciales. La Universidad de California contraatacó con otra demanda, argumentando que a su vez **AT&T** estaba usando el código de **BSD** sin respetar la licencia (la nota de reconocimiento a la universidad en la documentación y publicidad).

Después de algunas batallas legales, el caso fue resuelto fuera de la corte. Los abogados de Berkeley probaron que gran parte del código en disputa originado dentro de **BSD**, no era el original del **Unix** de **AT&T**. Sólo una docena de los archivos fueron propiedad de **AT&T**, mientras que el resto del sistema operativo perteneció al **CSRG** y sus contribuidores. **AT&T** tuvo

al final que remover la linea original del derecho de autor de Berkeley de su archivos que éste tomó de la **CSRG** y después de un tiempo finalmente liberó su **Unix System V**. Mientras tanto, el **CSRG** removió los archivos disputados y liberó a **BSD 4.4-Lite2** como una completa colección del código de **CSRG** completamente libre de cualquier derecho de autor de **AT&T**.

El código **BSD 4.4-Lite2**, también conocido como **Lite 2**, es el antecesor de todos los modernos sistemas **BSD**. Su contenido no fue útil en su presentación directa, por lo que necesitó de algunos aditamentos para funcionar. Varios grupos de programadores, tales como **BSDI**, el proyecto **NetBSD**, el proyecto **FreeBSD** y **OpenBSD** (como un descendencia directa de **NetBSD**) integraron a **Lite 2** sobre ellos mismos para hacer éste código útil y lo pudieran mantener dando como resultado que el desarrollo de **Lite 2** fuese manejado de manera independiente.

Los sistemas **BSD** hoy en día, han ganado el reconocimiento de ser sistemas maduros, robustos y muy estables. Cada uno posee su propio núcleo, cada sistema tiene su propio equipo que se encarga de él. Sin embargo, cuando se aclaró su situación legal, **Linux** era ya el sistema libre más popular. No obstante, éstos sistemas **BSD** siguen vigentes, están creciendo con más usuarios y siguen mostrando más funcionalidades.

El sistema **NetBSD** mantiene como prioridad fundamental que su distribución funcione sobre el mayor número de plataformas como sea posible, y hoy es capaz de comportarse exactamente igual sobre **PDA's** hasta grandes servidores. De ésta característica su lema: ***"Of course it runs NetBSD"***. Resulta una excelente elección para utilizar viejo hardware no Intel.

El sistema **FreeBSD**, en principio optó por concentrarse en los procesadores *x86* pero ahora funciona también sobre *alpha, amd64, sparc* y otros más. Es un sistema operativo sólido y en extremo eficiente, que obtiene el máximo rendimiento de la máquina. Otro de los objetivos iniciales era hacerlo más accesible a usuarios menos técnicos, y de hecho es el **BSD** libre con más usuarios en comparación con los demás proyectos **BSD**.

El sistema **OpenBSD** tiene como meta la seguridad y la integridad del código. El sistema combina el concepto de código abierto y una revisión rigurosa de sus archivos fuente que dan como fruto un sistema muy concreto. Es

elegido por instituciones preocupadas por la seguridad como bancos, entidades de cambio y departamentos de gobierno en el mundo. También funciona en una gran variedad de plataformas como **NetBSD**.

Algunos derivados de **BSD** son **MAC OS X** y **BSD/OS**. El **Mac OS X** es el actual sistema operativo de **Apple** perteneciente de la familia **Unix**. Sistema en donde el gestor de ventanas **X11**, ha sido sustituido por otro denominado **Aqua** desarrollado completamente por **Apple**. Su núcleo **BSD** se llama **Darwin** distribuido bajo la licencia **APSL**. Una licencia **Open Source** y **Software Libre** en sus últimas versiones. Sin embargo, las capas superiores del sistema, entre ellas el gráfico en general es código cerrado.

El sistema **BSD/OS** es un producto comercial de código cerrado producido por Wind River con gran parecido a los **BSD** de código abierto. El sistema está basado en **386BSD** y **4.4BSD**. Tiene la característica de poseer controladores para piezas de hardware de compañías que no liberan las especificaciones sobre sus productos.

Otro derivado que merece la pena mencionar es **PC-BSD**, el cual está basado en **FreeBSD**. El proyecto **PC-BSD** tiene como meta ser fácil de instalar y ser usado como sistema operativo de escritorio. Para lograr ésto, cuenta con una instalación gráfica que permitirá a los usuarios principiantes de **Unix** instalarlo y conseguir fácilmente que funcione.

El sistema **PC-BSD** es bueno, pero posee características que se podrían mejorar. Algunas de ellas, debido a que a sus desarrolladores les es difícil ver, como es el caso de la implementación de sus aplicaciones *.pbi*. Otro hecho es que no cubren a tope la gran mayoría de hardware que se puede encontrar. Es por eso que instalando y estudiando **PC-BSD** nos lleva a pensar sobre una mejorada distribución **BSD**.

Nuestro objetivo es utilizar los conceptos de básicos de la idea original de **PC-BSD**, las ideas del concepto de aplicaciones de **MAC OS X** (por su simplicidad) y tomando como núcleo del sistema a **OpenBSD** para intentar crear una distribución **BSD**. El nuevo sistema buscará integrar al entorno gráfico **KDE** con **OpenBSD** a un nivel más elevado al actual. Nuestro enfoque también dará oportunidad al usuario final de conocer más a los sistemas **BSD**. En donde se resalta el hecho de poder tener acceso al sistema opera-

tivo más seguro del mundo hasta nuestros días.

En el Capítulo 2, se habla de manera general sobre el sistema operativo **OpenBSD**. Se muestra su filosofía, fortalezas, su *árbol de puertos* y su seguridad entre otras cosas. Por último, se termina hablando sobre algunas de las diferencias más notables entre **FreeBSD** y **OpenBSD** que nos ayudaran a entender a éste tipo de sistemas.

En el Capítulo 3, se muestra el caso de estudio **PC-BSD** como punto de partida para el desarrollo de la nueva propuesta **BSD** mostrando sólo las características generales del sistema como lo son las modificaciones hechas a **FreeBSD** por su grupo de desarrollo y su sistema de aplicaciones **PBI**.

En el Capítulo 4, se muestra la idea de **PC-OpenBSD** como una mejor distribución **BSD** definiendo los conceptos y rasgos que se esperan dentro del sistema en su versión inicial. Aquí se presentan las ideas tomadas de **PC-BSD** y el concepto de aplicaciones de **MAC OS X** sobre **OpenBSD** integrado a **KDE**.

Dentro del Capítulo 5, se presentan las conclusiones sobre los principales aspectos y debilidades encontradas en la propuesta **BSD**. Así como posibles trabajos a futuro basados en el sistema encontrados al final de su desarrollo.

Por último, como una recomendación al lector, se espera que tenga conocimientos básicos sobre los sistemas operativos **BSD**, debido a que éste trabajo expone mejoras al sistema **OpenBSD** (con la integración de **KDE**) y no se enseña a usarlo, instalarlo o a configurarlo puesto que ese tipo de información es disponible en la página oficial del proyecto[1] o en varios sitios de **Internet**.

[1] www.openbsd.org

Capítulo 2

Marco teórico

En éste capítulo se presenta el marco teórico en donde en particular se darán algunos conceptos que envuelven al sistema operativo **OpenBSD**. Se presenta un resumen sobre sus fortalezas y las principales características que éste envuelve. Para finalizar se hablará acerca de los aspectos más importantes que lo diferencían de **FreeBSD**.

2.1. El Sistema Operativo OpenBSD

El proyecto **OpenBSD** nace de los ideales de *Theo de Raadt* quien empezó como un desarrollador del proyecto **NetBSD** varios años atrás. Cuando el tuvo varios desacuerdos con muchos desarrolladores de **NetBSD**, acerca de como el sistema debería ser desarrollado. *Theo de Raadt* abandona el proyecto y funda **OpenBSD** llevándose algunos desarrolladores que trabajaban con él.

Desde la fundación de **OpenBSD**, su equipo de desarrollo rápidamente estableció una identidad propia. Se identificaron como un grupo enfocado a la seguridad, siendo su sistema uno de los mejores **BSD** de código abierto hoy en día. El equipo introdujo varias ideas sobre los sistemas operativos de código abierto que existen en el mundo. Una de ellas es el acceso público a los repositorios del sistema vía **CVS**[1].

[1]El ***Concurrent Versioning System (CVS)***, es una aplicación informática que implementa un sistema de control de versiones, el cual mantiene un registro de todo el trabajo y los cambios en los archivos que forman un proyecto.

El sistema está basado sobre una fanática atención a seguridad, rectitud, usabilidad y libertad. Mientras que algunos otros **BSD** se enfocan en diferentes objetivos, **OpenBSD** se esfuerza por ser un sistema operativo en extremo seguro.

La interfaz gráfica de **OpenBSD** se basa en el ***X Window System (X11)***. Dispone de todos los gestores de ventanas habituales, como **Enlightenment**, **WindowMaker**, **GNUStep**, entre otros. También cuenta con los entornos gráficos como **KDE** y **GNOME** con todas sus aplicaciones. Además, de las aplicaciones disponibles para **OpenBSD**, se pueden ejecutar binarios compilados para **Linux**, **FreeBSD** y otros sistemas mediante una capa de compatibilidad. Normalmente no hay una diferencia de velocidad apreciable con la ejecución nativa de las aplicaciones en sus sistemas originales.

Actualmente el proyecto **OpenBSD** incluye todo el código actual del sistema operativo. Se cuenta con un manual en línea, se permite el acceso al ambiente de desarrollo, continuamente hace auditoría a varias de las licencias de software y ya cuenta con una importante comunidad de usuarios, desarrolladores y contribuidores.

2.1.1. Fortalezas de OpenBSD

Las principales fortalezas para proponer a **OpenBSD** como un excelente sistema operativo y el origen de su fama se resumen en los siguientes puntos:

Portabilidad

El sistema operativo **OpenBSD** está diseñado para correr sobre una gran variedad de procesadores y plataformas de hardware. Estas plataformas incluyen: **Intel PC** (*80386* y compatibles), **Alpha**, **Macintochs** (modelos **PowerPC** y *68000*), casi todas las máquinas de **Sun**, entre otras. En general el sistema nos da la oportunidad de poder correrlo sobre cualquier computadora que uno disponga.

Poder

El poder de **OpenBSD** radica en que puede correr sobre hardware que ha sido obsoleto por más de 10 años. Esta no es una decisión deliberada de su diseño, se debe a que éste hardware era popular cuando **OpenBSD** empezaba a desarrollarse. Es por eso que los desarrolladores tratan de mantener la velocidad y compatibilidad que más se pueda sobre cualquier hardware.

El sistema también da la posibilidad de brindar mayor poder de cómputo posible cuando uno corre aplicaciones (por el hecho que al final la gente usa aplicaciones y no sistemas operativos). Esto significa que un sistema con un giga en disco duro y un procesador *486* puede todavía ser un sólido servidor Web cuando uno instala **OpenBSD** en ésta máquina. Todo ello se debe a que un sistema operativo con menor huella da mayor explosión en el rendimiento del hardware.

Documentación

La documentación de **OpenBSD** es buena, las páginas del manual de las librerías y las llamadas al sistema son extensivas e incluyen discusiones sobre su uso y seguridad, siempre y cuando sea comparado con los otros **BSD**. Si durante la auditoría del código fuente de **OpenBSD**, el equipo de desarrollo encuentra cualquier número de circunstancias en donde la gente ha usado las interfaces de las librerías como las páginas del manual lo dicen, pero la página está incorrecta. Se considera como un error muy serio y es tratado como si se tratase de un error crítico de seguridad.

Libre

El sistema siempre trata de mantener el espíritu de la original **licencia BSD**[2], haciendo que **OpenBSD** sea libre de uso para cualquier propósito. Uno tiene la posibilidad de tomar **OpenBSD**, usarlo como una herramienta de trabajo sobre una computadora, modificarlo, si es posible venderlo y no pagar nada a los desarrolladores del proyecto. Caso que no se da con otras licencias usadas en otros programas de **Software Libre** como por ejemplo

[2]La licencia ***Berkeley Software Distribution*** es la licencia otorgada a los sistemas **BSD**. Se distingue por tener menos restricciones como las otras licencias de **Software Libre**, estando muy cercana al dominio público.

Linux que usa la **licencia GPL**[3]. Por todo lo anterior, el sistema siempre contará con una licencia extremadamente libre y un código correcto como lo afirma el fundador del proyecto *Theo de Raadt*.

Correcto

Los desarrolladores de **OpenBSD** siempre están enfocados en implementar soluciones de manera correcta. Esto significa que ellos siguen los estándares como **POSIX**[4] y **ANSI**[5] en todas sus implementaciones. Ellos hacen que ésto sea una regla estricta que da como resultado la escritura de programas más confiables y seguros, siguiendo siempre las mejores prácticas de programación actuales.

Todos los programadores tienen en mente que los programas escritos correctamente son más confiables, predecibles y por tanto seguros. Muchos productores de **Software Libre** están satisfechos si sus aplicaciones compilan y hace el trabajo deseado. Sin embargo, pocas compañías de software dan a sus programadores tiempo para escribir código correctamente. El código del sistema ha sido hecho correctamente, por el hecho que sus desarrolladores se toman el tiempo necesario para lograr éste objetivo.

Seguridad

El sistema se puede dar el lujo de ser el más seguro del mundo. Tal afirmación puede sostenerla de una manera razonablemente hasta el momento. Pero una posición de este tipo siempre requiere una constante lucha por mantener, debido a la existencia de gente que trata de averiguar nuevas maneras de penetrar dentro de un sistema. Hecho que significa que las características

[3]La licencia ***General Public Licence*** es una licencia creada por la ***Free Software Foundation*** para proteger la libre distribución, modificación y uso del software. Intenta proteger al software de intentos de apropiación que restrinjan las libertades de los usuarios.

[4]**POSIX** es el acrónimo de ***Portable Operating System Interface***; la ***X*** viene de **Unix** como una señal de identidad de la ***Application Programming Interface (API)***. Es un conjunto de estándares de llamadas al sistema operativo definidos por ***The Institute of Electrical and Electronics Engineers (IEEE)*** y especificados formalmente en el **IEEE 1003**.

[5]El ***American National Standards Institute (ANSI)***, es una organización sin ánimo de lucro que supervisa el desarrollo de los estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

de hoy en día pueden ser un hoyo de seguridad mañana.

Por todo lo anterior, los desarrolladores de **OpenBSD** cada vez que aprenden sobre la existencia de nuevas clases de hoyos de seguridad o errores de programación. Se dan la tarea de escanear el código fuente por completo para evitar que ésta clase de problemas estén dentro del sistema.

2.1.2. El *kernel* de OpenBSD

El *kernel* en **OpenBSD** es `/bsd`. A éste se le conoce como la interfaz entre el hardware y el software de usuario. El *kernel* permite escribir datos a los dispositivos de disco y de red, da instrucciones al CPU, cambia de sitio a los bits dentro de la memoria y en general provee una razonable interfaz al hardware a través de llamadas al sistema para la aplicaciones del usuario.

En algunos sistemas operativos de código abierto (tales como **Linux** o **FreeBSD**), la reconstrucción del *kernel* es recomendado sobre cualquier número de ocasiones. **OpenBSD** cae algunas veces en medio: puesto que el *kernel* estándar tiene la intensión de ser perfectamente útil sin ninguna modificación. Pero si uno requiere de herramientas para un mejor desempeño para cualquier proceso o alguna renovación deseada dentro del sistema, siempre se tiene acceso al código fuente por completo.

El *kernel* en el sistema es el primer programa que corre cuando éste inicia y es el responsable de todos los mensajes que se muestran antes de empezar a ejecutarse el proceso `init`[6]. Los mensajes de `init` se diferencian por tener un color diferente a los mensajes arrojados por los demás procesos.

El *kernel* de **OpenBSD** (como todos aquellos sistemas basados en **BSD**), incluye una variedad de parámetros modificables. Estos parámetros son conocidos como controles del sistema o *sysctls*. Algunos *sysctls* son estáticos y pueden ser vistos pero no cambiados. Los *sysctls* en **OpenBSD** están contenidos dentro del un ***Management Information Base (MIB)*** en forma de un árbol que organiza la información en categorías y subcategorías. La información de los valores del sistema o *sysctls* pueden ser cambiados mediante un administrador de *sysctls* denominado en **OpenBSD** como `sysctl`.

[6]El proceso `init` en **OpenBSD** es el primero que se ejecuta cuando el sistema inicia.

El comando `sysctl` permite ver los valores por defecto de todas las variables del *kernel*. También puede cambiar el valor de aquellos que sean estáticos mediante el nombre de la variable y su nuevo valor. En el Cuadro 1, se puede ver el uso de `sysctl` para la búsqueda de la variable `kern.usermount`. Acción en donde se asigna el nuevo valor de 1 para permitir a los usuarios poder montar dispositivos en su directorio local.

```
# sysctl -a | grep kern.usermount
kern.usermount=0

# sysctl kern.usermount=1
kern.usermount: 0 -> 1
```

Cuadro 1: Búsqueda y modificación de la variable del *kernel* `kern.usermount` con `sysctl`.

Otra manera de alterar al *kernel* es mediante el comando `config`[7]. Tal comando permite crear un directorio de compilación del *kernel* desde un archivo de configuración especificado. Otra característica es la edición del binario del *kernel* (`/bsd`) de manera dinámica permitiendo habilitar, deshabilitar o modificar al *kernel* sin la necesidad de algún proceso de compilación todo hecho antes de iniciar el sistema.

2.2. El manejo del disco y el Sistema de Archivos

2.2.1. Nodos de los dispositivos

Un *Nodo de dispositivo* es un archivo especial que provee una interfaz lógica a una pieza de hardware, en donde se pueden realizar operaciones de lectura y escritura a través de comandos del sistema. Generalmente los nodos de los dispositivos están encriptados y varían su uso entre los sistemas operativos.

[7]Para más información de `config` puede consultar las páginas del manual del sistema.

El sistema operativo **OpenBSD** (al igual que **FreeBSD**) almacena todos los nodos de los dispositivos dentro del directorio `/dev`. En el Cuadro 2.1, se puede ver una lista de los dispositivos de disco dentro de **OpenBSD**.

Nombre del dispositivo	Descripsión
/dev/fd*	Disco floppy (bloque)
/dev/rfd*	Disco floppy "raw"
/dev/wd*	Disco IDE (bloque)
/dev/rwd*	Disco IDE "raw"
/dev/sd*	Disco SCSI (bloque)
/dev/rsd*	Disco SCSI "raw"
/dev/cd*	Dispositivo CD-ROM IDE o SCSI (bloque)
/dev/rcd*	Dispositivo CD-ROM "raw"

Cuadro 2.1: Lista de dispositivos de disco en OpenBSD.

Los nombres de los dispositivos en éste sistema tienen un número que indica a que instancia del dispositivo se está refiriendo. Por ejemplo, `/dev/wd0` es el primer disco disco IDE, `/dev/wd1` es el segundo y `/dev/cd1` es el segundo dispositivo CD.

En **OpenBSD** cada partición tiene una letra, la partición raíz es "a", la partición *swap* es "b" y el disco entero es "c". Cada una de éstas particiones tiene un nodo de dispositivo por separado, el cual está dado por la suma de la letra de la partición y el nombre del dispositivo. Por ejemplo, en **OpenBSD** `/dev/wd0a` es la partición raíz del primer disco duro **IDE**, y `/dev/sd3b` es la partición *swap* del cuarto disco **SCSI**.

Otro tipo de dispositivos dentro de **OpenBSD** son aquellos con nombre *raw*. Los dispositivos *raw* son una manera diferente de acceder a los mismos nodos de los dispositivos. Los programas son los que generalmente eligen la manera de acceso, ya sea a través de dispositivos estándar o a través de un nodo de dispositivo *raw*.

2.2.2. La tabla de Sistemas de Archivos: `/etc/fstab`

La tabla de los sistemas de archivos en **OpenBSD** es `/etc/fstab`. Dicha tabla lista todos los dispositivos de disco configurados para su uso dentro del sistema. Aquí se muestra información sobre que sistema de archivos están montados. También se presentan opciones especiales usadas para montar tales dispositivos. En el Cuadro 2, se muestra un ejemplo del contenido de `/etc/fstab` en donde se aprecian los sistemas de archivos y algunas opciones para cada uno de ellos.

#Device	MountPoint	Fstype	Options	Dump	Pass#
/dev/wd0a	/	ffs	rw	1	2
/dev/wd0g	/home	ffs	rw,nodev,nosuid	1	2
/dev/wd0d	/tmp	ffs	rw,nodev	1	2
/dev/wd0f	/usr	ffs	rw,nodev	1	2
/dev/wd0e	/var	ffs	rw,nodev,nosuid	1	2

Cuadro 2: Ejemplo del contenido de fstab.

La información dentro de `/etc/fstab` se divide en campos, en donde el primero da el nodo del dispositivo con alguna partición en especial. El segundo campo indica el punto de montaje, el directorio en donde el sistema de archivos se encuentra. El tercer campo indica el tipo de sistema de archivos usado. En las particiones **OpenBSD** se usa el tipo ***Unix Fast File System (ffs)***. Otros tipos son *msdos* para particiones **FAT**, ***mfs (Memory File System)*** y **cd9660** para dispositivos CD entre otros más. El cuarto campo muestra la opciones de montado para éste sistema de archivos. El quinto campo es un parámetro que será usado por `dump`[8]. El último campo, es el *pass number*, valor que dice a `fsck`[9] en que momento debe ser examinado el sistema de archivos durante el proceso de inicio del sistema.

2.2.3. El sistema de archivos *Fast File System*

El sistema de archivos de **OpenBSD** algunas veces llamado **UFS**, es descendencia directa del sistema de archivos incluido dentro de **BSD4.4**. En

[8]El comando `dump` examina el contenido en un sistema de archivos para determinar que archivos necesitan ser respaldados

[9]El comando `fsck` es un programa que checa la consistencia de los sistemas de archivos de los dispositivos declarados en `/etc/fstab`.

la actualidad hay las personas quienes trabajaban sobre el original **BSD**, quienes están haciendo mejoras al sistema de archivos (las cuales son frecuentemente importadas dentro de **OpenBSD**).

El sistema de archivos **FFS** está diseñado a ser rápido y confiable. Dentro de **OpenBSD** se ha configurado a ser extremadamente útil como sea posible para cada arquitectura, pero uno puede elegir el momento para ser optimizado. Dentro de **FFS** existen *bloques*, *fragmentos* e *inodos*.

Un archivo está formado por *bloques* y *fragmentos*. Un *bloque* son largos pedazos de datos mientras que los *fragmentos* son pequeños pedazos. Generalmente un archivo esta hecho con muchos *bloques* como sea posible y usa uno o dos *fragmentos* para el resto de los datos. Cuando un archivo crece, el sistema localiza bloques adiciones para tal información.

Los *inodos* contienen información acerca del archivo, éste incluye una lista de los *bloques* y *fragmentos* que el archivo usa, además de los permisos y tamaño de tal archivo. La información en los *inodos* es llamada *metadatos*, cuyo significado es *"datos acerca de los datos"* o *índices de inodos* de los archivos sobre un disco.

Al igual que en otros sistemas, las particiones **FFS** pueden ser tratadas de diferentes maneras dependiendo de como quieren accedidos los datos. La manera en que una partición es montada es llamado *tipo de montado*. Uno puede cambiar las opciones de montado en alguna partición estándar mediante la edición de algunos valores en las entradas del archivo `/etc/fstab`.

Las opciones de montado sobre las particiones **FFS** son las siguientes:

1. *Read-Only:* Opción que sólo permite visualizar el contenido de la partición y no da acceso de escritura.

2. *Read-Write:* Opción que permite las operaciones de lectura y escritura en una partición específica.

3. *Soft Update:* Algunas veces llamada *softdep*, organiza y prepara las escrituras al disco haciendo que los *metadatos* en el sistema de archivos continuen con su consistencia del disco.

4. *Synchronous:* Esta opción permite que la partición sea leída más rápidamente pero el proceso de escritura es mucho más lento de lo normal.

5. *Noexec:* Opción que previene la ejecución de binarios dentro de la partición especifíca.

6. *Nosuid:* Opción que deshabilita el comportamiento de programas con *setuid*.

7. *Noauto:* Opción que permite definir que particiones no deben de ser montadas automáticamente en el proceso de inicio del sistema. Es frecuentemente usada con dispositivos CD y floppies.

8. *No Access Time*: Opción que permite registrar la última vez que fue accesado algún archivo, no importando si fue ejecutado o leído por cualquier medio.

9. *No Device:* Opción que le dice al sistema a no interpretar cualquier nodo de dispositivo sobre el sistema.

Con las opciones descritas anteriormente uno puede cambiar o modificar las opciones de montado como se muestra en Cuadro 2, en donde se definen la opciones `rw, nodev` y `nosuid` para la partición `/dev/wd0g` destinada a `/home`.

2.2.4. Sistemas de Archivos Foráneos

El sistema operativo **OpenBSD** puede trabajar también con sistemas de archivos foráneos como lo son los medios de almacenamiento CD. Los sistemas de archivos soportados son los siguientes:

1. *ados:* Permite utilizar sistema de archivos de **AmigaDOS** dentro de dispositivos especiales en el sistema de archivos original.

2. *cd9660:* Permite utilizar sistema de archivos **ISO 9660** el cual es utilizado por las unidades CD.

3. *ext2fs:* Permite utilizar sistema de archivos **ext2fs** usado en las particiones de **Linux**.

4. *msdos:* Permite utilizar sistema de archivos **FAT** usado principalmente cuando se trabaja con memorias **USB** y floppies.

5. *nfs:* Permite al sistema preparar y acceder un sistema de archivos remoto.

6. *mfs:* Opción que permite construir un sistema de archivos virtual en memoria.

7. *ntfs:* Permite montar sistema de archivos **NTFS**. En **OpenBSD** sólo es soportando el proceso de lectura sobre éste tipo de sistema de archivos.

8. *udf:* Opción que permite trabajar con sistema de archivos **UDF** encontrado principalmente dentro de unidades DVD.

9. *procfs:* Opción que permite acceder al espacio de trabajo de los procesos.

Todos los tipos de sistema de archivos foráneos mencionados anteriormente, son accedidos mediante el uso del comando `mount`[10]. Su sintáxis es muy sencilla dentro del sistema. El comando se puede utilizar con muchas opciones dependiendo del tipo de sistema de archivos.

2.3. El *árbol de puertos y paquetes*

El *árbol de puertos y paquetes* de **OpenBSD** es un sistema de construcción de software diseñado para simplificar la configuración e instalación de software. Originalmente fue desarrollado por el proyecto **FreeBSD** y fue expandido dentro de **OpenBSD**.

Los *puertos* son instrucciones para compilar software sobre una particular versión de **OpenBSD** y los *paquetes* son simplemente *puertos* precompilados. Los *paquetes* se instalan rápidamente y pueden ser una excelente opción cuando se instala software que no necesita personalización. Por otro lado los *puertos* se instalan más lentamente, pero pueden ser fácilmente personalizados y optimizados para un ambiente en especial.

[10]Para más información del comando `mount` consultar las paginas de manual de **OpenBSD**

La idea básica del *árbol de puertos y paquetes* es muy simple: si el software debe ser modificado para correr sobre **OpenBSD**, entonces las modificaciones deben ser automatizadas. Dando por resultado que uno pueda usar las mismas instrucciones para instalar un programa precompilado en cualquier otro sistema **OpenBSD**.

2.3.1. El árbol de puertos

El *árbol de puertos* **OpenBSD** en general mantiene un completo registro de todo el proceso de instalación de software que se ha hecho. Ello libera al usuario de pensar en cómo instalar un programa y lo concentra ha hacer que el programa trabaje correctamente.

En la actualidad, **OpenBSD** ya cuenta con más de 4000 programas clasificados en diferentes categorías y subcategorías empleadas dentro del *árbol de puertos*. Generalmente, el *árbol de puertos* está ubicado en `/usr/ports`. En el Cuadro 3, se puede apreciar el listado de las categorías del directorio `/usr/ports` dentro del sistema.

`.cvsignore`	`chinese`	`inputmethods`	`plan9`
`CVS`	`comms`	`japanese`	`print`
`INDEX`	`converters`	`java`	`productivity`
`Makefile`	`databases`	`korean`	`russian`
`README`	`devel`	`lang`	`security`
`archivers`	`editors`	`mail`	`shells`
`astro`	`education`	`math`	`sysutils`
`audio`	`emulators`	`misc`	`telephony`
`benchmarks`	`games`	`multimedia`	`textproc`
`biology`	`geo`	`net`	`www`
`books`	`graphics`	`news`	`x11`
`cad`	`infrastructure`	`palm`	

Cuadro 3: Categorías del *árbol de puerto*s dentro de **OpenBSD**.

Dentro de todas las categorías del *árbol de puertos*, existen directorios que representan su jerarquía los cuales son los siguientes:

1. El directorio `CVS` cuenta con información del sistema sobre el control de revisión usado por **OpenBSD** para ver la versión actual del sistema.

2. El archivo `INDEX` contiene un lista de todos los *puertos*.

3. El archivo `Makefile` contiene instrucciones para construir o manejar la entera colección de *puertos*. Es usado generalmente cuando se construyen *paquetes* para una versión a liberar de **OpenBSD**.

4. El directorio `infrastructure` cuenta con herramientas para construir al *árbol de puertos*. Aquí se encuentran la configuraciones globales de los *puertos*, así como otras instrucciones requeridas durante el proceso construcción.

5. El archivo `README` cuenta con una breve introducción al *árbol de puertos* y apuntadores a varias páginas del manual que describen su funcionalidad.

6. Finalmente, el directorio `distfiles` cuenta con el código fuente para varios *puertos*. Generalmente, al principio está vacío, pero cuando el usuario instala un *puerto*, **OpenBSD** descargará los archivos de **Internet** en éste directorio.

Si uno opta por instalar aplicaciones mediante ésta vía, el sistema requiere estar conectado a **Internet** para poder ejecutar las operaciones `make search key, make, make install y make clean`.

El comando `make search key` o también `make search name` es usado para buscar algún *puerto* en especial dentro del *árbol de puertos*.

Para poder usar las demás operaciones, se requiere por fuerza estar dentro del directorio del *puerto* a instalar. Para compilar se hace uso del comando `make`. Luego de ello, si uno desea instalar el *puerto* dentro del sistema se utiliza el comando `make install`. Si uno desea eliminar todos los objetos que fueron creados durante la compilación del *puerto*, se usa al comando `make clean`. Si uno desea compilar, instalar y limpiar los objetos ya usados en una misma instrucción puede usar el comando `make install clean`.

Si durante el proceso de instalación el *puerto* tiene dependencias, el sistema los compilará e instalará a ellos también. Si uno desea eliminar tal *puerto*, se puede hacer uso del comando `pkg_delete` o con el uso de `make deisntall`.

2.3.2. Los *paquetes*

Los *paquetes* son software precompilado para alguna particular versión y arquitectura de **OpenBSD**. Al menos cada *puerto* de **OpenBSD** está disponible como un *paquete* salvo algunas piezas de software que no pueden ser redistribuidas por sus restricciones. Así que tales aplicaciones sólo estan disponibles como *puertos*. Los *paquetes* cuentan con la extensión **.tgz* y están disponibles en el CD-ROM de **OpenBSD** o vía **FTP**.

Para poder manejarlos se emplean los comandos del sistema `pkg_add`, `pkg_delete` y `pkg_info`.

El comando `pkg_add` instala un *paquete* y todos los que éste requiera. En donde cada *paquete* instalado crea un directorio con su nombre en `/var/db/pkg`. Si uno no cuenta con los *paquetes* en la máquina local, el comando usará el valor de la variable `PKG_PATH` para que los *paquetes* sean descargados desde el servidor **FTP** definido en tal variable. Por ésta razón, muchos usuarios de **OpenBSD** definen el valor ésta variable de entorno dentro de los archivos `.profile` o en `.xsession` dentro de la raíz del directorio del usuario `root`. En el Cuadro 4, se puede ver la declaración de ésta variable para el caso de una máquina con arquitectura *i386* que tiene instalado la version 4.1 del sistema.

```
export PKG_PATH=ftp://ftp.openBSD.org/pub/OpenBSD/4.1/
packages/i386
```

Cuadro 4: Declaración de la variable `PKG_PATH`.

El comando `pkg_info` muentra todos los *paquetes* instalados en el sistema, y el comando `pkg_delete` desinstala a los *paquetes* que le son pasados como parámetros. Con la opcion `-f dependencies` se le dice a `pkg_delete` que elemine primero aquellos *paquetes* que dependan del *paquete* que será eliminado.

2.4. Características adicionales de seguridad

La primera y mejor línea de defensa contra intrusos es mantener al sistema con las actualizaciones del día. Tales actualizaciones se producen por el

trabajo continuo que los miembros del proyecto de **OpenBSD** realizan auditando el código fuente del sistema, tratando de encontrar formas de poder dañarlo. Todo ese trabajo les ha dado la reputación que **OpenBSD** tiene de ser el sistema más seguro del mundo que no viene de la nada, sino a través de las características de seguridad que el grupo de desarrollo ha incluido dentro del sistema

2.4.1. *Checksums*

Un *checksum* es el resultado de tomar un pedazo de datos (tal como un archivo) para aplicarle un algoritmo matemático y de cómputo para producir una cadena de caracteres. Si el archivo original cambia de cualquier manera, el *checksum* muy posiblemente también lo hará.

Los *checksums* proveen una manera de verificar la integridad de los datos y muchos sitios de distribución de software (incluyendo los servidores **FTP** de **OpenBSD**) incluyen *checksums* para los archivos que ellos almacenan. Por tal motivo, toda la arquitectura de sus directorios incluyen archivos llamados **MD5**. Archivos que incluyen *checksums* para cada uno del los archivos de la distribución.

El sistema incluye por defecto con herramientas para calcular *checksums* para **MD5, SHA-1** y **RMD-160**. El más popular es el tipo **MD5**, pero las otras son perfectamente validas. Cada algoritmo es muy diferente uno de otro, al igual que sus resultados. Uno debe usar la herramienta apropiada para el tipo de *checksum* que se intente verificar. En el Cuadro 5, se muestra el uso de comando `md5` para calcular el *checksum* del archivo `ports.tar.gz` que contiene el *árbol de puertos* del sistema.

```
# md5 ports.tar.gz
MD5 (ports.tar.gz) = f349c80fef2e857c05f46052701e7cb3
```

Cuadro 5: Cálculo del *checksum* de `ports.tar.gz` con el comando `md5`.

2.4.2. Banderas en los archivos

El sistema de permisos de **Unix** es estándar dentro de las varias versiones de los sistemas **BSD**, pero **OpenBSD** extiende el esquema de permisos con banderas en los archivos.

Las banderas en los archivos trabajan directamente con los permisos para aumentar la seguridad del sistema y los usuarios mediante el cambio de la manera en que un archivo es accesado. Las banderas pueden hacer que el archivo sea incambiable, hacer que en éste no se le puedan remover datos y sólo tenga opción del poder agregar nuevos datos. Tales banderas trabajan en combinación con los niveles de seguridad, puesto que los dos sistemas se relacionan mutuamente.

La bandera *sappnd*

La bandera del nivel de seguridad de sólo ingreso *(append only)*, sólo puede ser puesta o removida por el `root`. Los archivos con ésta bandera permiten el ingreso de datos pero no permiten que dentro de ellos se presente el removido o el cambio de datos. Su uso es principalmente por los archivos de *log* del sistema. La bandera no puede ser removida cuando el sistema ésta corriendo en un nivel de seguridad 1 o mayor.

La bandera *schg*

La bandera de nivel de seguridad inmutable sólo puede ser puesta o removida por el `root`. Los archivos con ésta bandera no pueden ser cambiados de ninguna forma (no se permite su edición, no pueden ser removido y no pueden ser reemplazados). Básicamente el *kernel* proviene el uso de las operaciones no permitidas que se intenten hacer al archivo anulándolas por completo. La bandera no puede ser removida cuando el sistema está corriendo en un nivel de seguridad 1 o mayor.

La bandera *uappnd*

Sólo el dueño o el `root` pueden poner al usuario la bandera *append only*. Como en la bandera `sappnd`, la bandera de usuarios *append only* permite que sólo le sean agregados datos, pero no puede ser editado o removido. Esto es muy útil para los *logs* de programas personales y significa que los usuarios

pueden preservar sus archivos vitales de un borrado accidental. El usuario o el `root` pueden remover ésta bandera en cualquier momento.

La bandera *uchg*

Sólo el dueño o el `root` pueden poner al usuario la bandera inmutable. Como la bandera *schg*, la bandera de usuario inmutable evita que el usuario pueda realizar el intercambio de archivos. El `root` puede remover ésta bandera así como el usuario en cualquier nivel de seguridad.

2.4.3. Niveles de seguridad

Los niveles de seguridad en **OpenBSD** son una de las características más importantes con las que cuenta. Se dividen en 4 niveles: -1, 0, 1, y 2, en donde -1 es el menor nivel de seguridad y 2 el mayor.

El nivel de seguridad dentro del sistema por defecto es 1. El nivel de seguridad, se define en el archivo `/etc/rc.securelevel`. Si uno quiere cambiar de seguridad puede modificar tal archivo y reiniciar el sistema.

Si uno quiere cambiar el nivel de seguridad sin rebootear el sistema, puede hacer uso del comando `sysctl` como se muestra en el Cuadro 6, en donde el nivel de seguridad pasa de -1 a 1.

```
# sysctl -w kern.securelevel=1
kern.securelevel: -1 -> 1
```

Cuadro 6: Cambio de nivel de seguridad con el comando `sysctl`.

Un hecho que cabe resaltar es que el nivel de seguridad no puede bajar cuando el sistema esta en marcha, por el hecho haber intrusos que quieran bajar el nivel de seguridad.

Nivel de seguridad -1

El nivel de seguridad -1 provee seguridad adicional al *kernel*. Es recomendado cuando alguien ésta aprendiendo a usar el sistema y hace experimentos con algunas características y configuraciones. En éste nivel las características estándar de los **Unix** tales como los permisos son completamente funcionales.

Nivel de seguridad 0

Este nivel de seguridad sólo es usado cuando el sistema inicia por primera vez. Cuando el sistema alcanza el modo *multiusuario* el nivel de seguridad es puesto automáticamente al nivel de seguridad 1.

Nivel de seguridad 1

Es el nivel de seguridad por defecto de **OpenBSD**. Sus principales puntos son los siguientes:

1. Nadie puede escribir a los dispositivos `/dev/men` y `/dev/kmen`. Muchos de los riesgos de seguridad provienen por he hecho de poder escribir a tales dispositivos.

2. Los dispositivos de disco *raw* de todos los sistemas de archivos son sólo de lectura. Ello implica que todos los accesos a disco deben de ser hechos con los dispositivos estándar y los programas sólo pueden accesar a sistemas de archivos montados a través de los dispositivos estándar sin cambiar las operaciones que realicen.

3. Las banderas *schg* y *sappnd* no pueden ser removidas. Uno necesitaría reiniciar el sistema y entrar en *sigle-user* para quitar tales banderas.

4. Los módulos del *kernel* no pueden ser cargados o descargados. Tal carecterística que no afecta en casi nada a las necesidades del administrador del sistema.

Nivel de seguridad 2

El nivel de seguridad 2 es el más alto que ofrece el sistema. En general deshabilita una variedad de características que pueden ser requeridas durante un proceso de mantenimiento normal.

Usando un nivel de seguridad como éste, hace que el sistema sea menos flexible pero evita muchos cambios no autorizados. El nivel se seguridad 2 incluye todos los efectos del nivel de seguridad 1 más los siguientes cambios adicionales:

1. Los dispositivos de disco *raw* son siempre sólo de lectura.

2. El reloj del sistema no puede ser actualizado hacia atrás.

3. El comando `pfctl` no puede alterar las reglas de ***Package Filter (PF)*** o **Network Adress Traslation (NAT)**.

4. El ***DDB kernel debugger (depurador del kernel)*** no puede cambiar los valores usados por `sysctl`.

El nivel de seguridad 2 puede ser visto irrelevante para nuevos administradores del sistema. Pero las características que éste ofrece pueden ser muy importantes. Si principal objetivo es evitar que los intrusos tengan herramientas para poder atacar al sistema. Puesto que varios de los ataques a los que se recurren, se dan través del uso directo de los dispositivos *raw*. La meta de éste tipo de ataques, es hacer cambios en el sistema de archivos sin la necesidad de poner atención en las banderas de permisos en los archivos. Cambio que no pueden hacer en éste nivel de seguridad, puesto que los dispositivos *raw* son sólo de lectura.

2.4.4. Systrace

Otra carecterística de seguridad importante del sistema es `systrace`. La herramienta `systrace` funge como un administrador de acceso a las llamadas al sistema. Con `systrace`, un administrador puede decir que llamadas del sistema pueden ser hechas por los programas y la manera en que éstas deben ser hechas. El apropiado uso de `systrace` puede enormemente reducir los riesgos de la explotación de programas por intrusos. Las políticas del `systrace` pueden confinar a los usuarios de una manera completamente independiente a los permisos de **Unix**. Uno puede siempre definir los errores que las llamadas del sistema regresan cuando su acceso es denegado. Permite a los programas fallar de una manera deseada, pero requiere de un entendimiento práctico de las llamadas del sistema, saber sobre qué programas deben de trabajar correctamente y lo más importante como éstas cosas interactuan con la seguridad.

2.5. Diferencias entre OpenBSD y FreeBSD

Tanto **OpenBSD** como **FreeBSD** descienden directamente de **BSD4.4**. Es por ello que varias de sus operaciones internas y manejo de archivos es

muy similar. Pero aun así existen diferencias significativas las cuales se listan algunas de ellas a continuación:

1. El sistema **OpenBSD** puede alterar su *kernel* desde su *prompt* de inicio con ayuda de su comando `config`, función que **FreeBSD** no tiene puesto que si éste necesita agregar o configurar una pieza de hardware no detectada correctamente (pero si soportada) es necesaria la compilación de su *kernel*.

2. Por seguridad **OpenBSD** no cuenta con la opción de dar a los usuarios el poder manejar directamente los dispositivos *atapicam*. Opción que permite (en el caso de los dispositivos CD) ver si algún dispositivo está listo o no. El comando que permite éste tipo de interacción entre los dispositivo *atapicam* dentro de **FreeBSD** es `camcontrol`.

3. Los dispositivos en **OpenBSD** no tienen el mismo nombre dentro de **FreeBSD**. Otra diferencia en lo referente a los dispositivos es que en **FreeBSD** no existen los dispositivos *raw*.

4. Aunque **FreeBSD** intenta tener los niveles de seguridad de **OpenBSD**, tales niveles no son tan efectivos. Otras opciones de seguridad no presentes en **FreeBSD** son: el administrador de acceso de las llamadas al sistema `systrace`, las banderas de los archivos y usuarios entre otras características de seguridad (como la encriptación) nativas de **OpenBSD**.

5. En general **FreeBSD** soporta menos plataformas de hardware que **OpenBSD** por el resultado de tener **OpenBSD** la descendencia directa con **NetBSD**[11].

Además de la lista mostrada anteriormente, existen más diferencias que involucran la manera de hacer algunas operaciones en ambos sistemas. Por ejemplo, para el caso del manejo de imagenes de disco en **OpenBSD** se utiliza `vnconfig` y en **FreeBSD** `mdconfig`. Pero al fin y al cabo los dos tienen la opción de poder trabajar con éste tipo de archivos.

[11]Ver el artículo *Installing **BSD** Operating Systems On IBM Netvista S40* en http://www.daemonnews.org/

Capítulo 3

El caso de estudio de PC-BSD

3.1. PC-BSD como sistema de escritorio

El sistema **PC-BSD** en esencia es **FreeBSD** con algunas modificaciones significativas. Cambios que van desde alteración parámetros en la configuración del *kernel*, la alteración de los valores de algunas variables del sistema, la inclusión de algunos binarios. Además, se incluyen varios *scripts* con diferentes funciones que van desde el tener un asistente que permite la instalación de programas, hasta el poder controlar el comportamiento de algunos servicios del sistema como la autodetección de dispositivos.

Las características más importantes del sistema se mencionan a continuación:

1. El sistema cuenta con un instalador gráfico escrito en el lenguaje **QT**.

2. Por defecto el usuario cuenta con un escritorio **KDE** ya configurado y personalizado. Así el usuario nunca tiene que hacer cambios adicionales para poder usar el sistema e iniciar sesión después de su instalación. Todo lo anterior se da porque por la existencia de un esqueleto en donde se definen que archivos debe tener un usuario recién creado.

3. Un usuario puede realizar operaciones del *superusuario* puesto que éste puede disponer del comando `su`, ya sea para administración del sistema o simplemente para hacer un cambio significativo en el mismo.

4. Los usuarios también pueden hacer uso de algunos dispositivos del sistema de manera gráfica. Algunos de ellos son las memorias **USB**, unidades CD tanto de audio como de datos, floppies, entre otros más. También cualquier usuario es capaz de montar y desmontar un dispositivo en algún directorio en donde él sea dueño.

5. Los usuarios tienen servicios predefinidos para poder compartir archivos y carpetas entre otras computadoras gracias al Servicio de **Samba**.

6. Los usuarios desde de **KDE** pueden configurar y hacer uso de cualquier tipo de impresora que sea reconocida por el sistema. Todo ello gracias a **CUPS**, no importando si la interfaz es vía **USB**, mediante cable paralelo, interfaces de red e incluso es posible acceder a impresoras compartidas en red con ayuda del servicio de **Samba**.

7. El proyecto **PC-BSD** tiene aplicaciones las cuales son fáciles de instalar. Los usuarios tienen varios tipos de aplicaciones que pueden descargar directamente desde **Internet** como lo es el *paquete* de oficina **OpenOffice** hasta herramientas de desarrollo como **JavaBeans**.

3.2. El disco de Instalación de PC-BSD

El disco de instalación en esencia es un **Live CD** basado en **FreeSBIE**[1]. A tal disco se le hicieron algunas modificaciones en donde se incluyeron algunos binarios y *scripts* que generan el proceso de instalación.

Un análisis de manera general del **Live CD** de instalación se da en los siguientes puntos:

1. Se tiene instalado al *paquete* `dialog`[2]. El uso de éste es principalmente desde los *scripts* del proceso de bienvenida del **Live CD**.

2. El *superusuario* `root` por defecto no tiene definido una contraseña, lo que posibilita su entrada al sistema sin ningún tipo de autentificación.

[1] **FreeSBIE** es un **Live CD** vivo basado en **FreeBSD** para arquitecturas *i386*, el cual cuenta con **XFCE** como entorno gráfico.

[2] El *paquete* `dialog` que permite mostrar cuadros de diálogo desde *scripts* para ser mostrados en consola.

3. El archivo `/etc/ttys`[3] define que la primera consola virtual sea una sesión del `root`.

```
ttyv0  "/usrl/libexec/getty root" cons25 on  secure
```

4. La última línea del archivo de configuración de `.login` (de usuario `root`), define la ejecución de un *script* llamado `PCBSDStart.sh` ubicado en `/root`. Tal archivo que contiene un menú con las siguientes opciones:

   a) *Install:* Iniciar el instalador gráfico(X ,Y).

   b) *Xreset:* Reiniciar al servidor **X11** al controlador genérico **VESA**.

   c) *Utility:* Utilidades del sistema.

   d) *Raid:* Soporte para **RAID**.

   e) *Reboot:* Reiniciar el sistema.

   Si uno procede a seleccionar *Install* se mandará a ejecutar a `startx`. Instrucción que por definición lee el archivo de configuración del `root` `.xinitrc` cuyo contenido más importante son las siguientes líneas:

```
/usr/local/pcbsd/bin/PCInstall
shutdown -r now
```

   La primera línea implica ejecutar al binario `PCInstall`, el cuál es programa de instalación gráfico de **PC-BSD**. Al terminar la ejecución de éste binario se ejecuta la línea siguiente indicando el reinicio de la máquina.

   Si la opción fue *Reset* se configurará la resolución gráfica del servidor **X11** para la instalación usando el controlador genérico *VESA*. El resultado será una resolución de *800x600* píxeles.

   La opción *Utility* manda a ejecutar a otro *script* llamado `PCBSDUtil.sh` el cual tiene por propósito mostrar un submenú conformado por la siguientes opciones:

[3]El archivo `/etc/ttys` contiene la configuración de las consolas virtuales del sistema.

a) *fdisk:* opción que corre `fdisk`[4] para su uso de manera manual.

b) *shell:* opción que inicia la ejecución del comando `sh` como *shell* de emergencia.

c) *chroot:* opción para cambiar el directorio de `root` en las particiones.

d) *exit:* opción para salir de éste menú, el cual hará un regreso al menú principal de instalación de **PC-BSD**.

Por último la opción *Reboot* reiniciará el sistema.

5. El archivo de `.xinitrc` define que se cargue al `PCInstall`.

6. El directorio del instalador gráfico está situado en `/usr/local/pcbsd`. El cual contiene varios archivos como lo son *scripts*, imágenes y al binario de instalación. Los directorios que conforman al instalador son los siguientes:

   a) `bin`: contiene al binario `PCInstall` escrito en el lenguaje de programación **C++** junto con la extensión **QT**.

   b) `images`: contiene todas las imágenes usadas por `PCInstall`.

   c) `scripts`: dentro de él radican todos los *scripts* que se mandan a ejecutar por el binario `PCInstall`.

   d) `LANGS`: contiene todos los archivos de traducción de la aplicación `PCInstall`, los cuales son cargados dinámicamente cuando el usuario selecciona algún idioma durante la instalación.

7. El archivo comprimido `PCBSD.tgz` tiene como contenido todo el sistema, el cual será descomprimido en el disco duro destinado para **PC-BSD**.

En teoría todo lo descrito es replicable para crear un CD de instalación de **OpenBSD** puesto que en él también existe el *paquete* `dialog` y la programación de *scripts* es un punto básico en éstos sistemas. La única observación sobre el instalador gráfico con el que cuenta **PC-BSD**, se centra en que algunas de las operaciones que ejecuta su instalador no están soportadas en **OpenBSD**. Es por eso que se tratará de dar una opción de instalación entendible y fácil para los usuarios.

[4]El comando `fdisk`, es un programa que permite dividir en forma lógica un disco duro.

3.3. Análisis de la características de PC-BSD

3.3.1. Análisis del *kernel*

La configuración del *kernel* de **PC-BSD** es un punto importante dentro de ésta distribución. Su configuración está orientada a correr principalmente en máquinas pertenecientes a las arquitecturas *i386* con uno o más procesadores. Su comportamiento es similar al *kernel* `GENERIC` que viene por defecto, pero se distingue por contar con una modificación que se genera por la necesidad de tener al 100 % la funcionalidad del comando `camcontrol`[5]. El funcionamiento de `camcontrol` en un *kernel* genérico, sólo proporciona información sobre dispositivos de almacenamiento masivos como lo son las unidades **USB**. Sin embargo, con la adición de la opción correspondiente al *kernel*, éste también es capaz de proporcionar información sobre los dispositivos CD.

Para tener esta característica se requiere agregar en las opciones de los dispositivos **ATA** y **ATAPI** del archivo de configuracion del *kernel* la línea:

```
option  device  atapicam   # Atapi CAM Support
```

Con los anterior, el servicio de detección de unidades CD proporcionado por **PC-BSD** (independiente del servicio **HAL**) podrá trabajar adecuadamente.

3.3.2. Análisis del directorio `/etc`

El comportamiento del sistema se centra por el contenido de los archivos de configuración pertenecientes a `/etc`. Aquí se encuentran datos como lo son: el estado de las variables del *kernel*, la configuración de las interfaces de red del sistema, los archivos de bases de datos de los usuarios y grupos, entre otros.

A continuación se presentan y describen las modificaciones hechas a los archivos de configuración más característicos de éste directorio:

[5]La utilidad `camcontrol` está diseñada para proveer a los usuarios una manera de control de los dispositivos **SCSI** y **ATAPI** de **FreeBSD** como lo son las memorias **USB** y los discos CD.

1. El archivo `/etc/crontab`[6] fue modificado por el uso del *paquete* `Anacron`.

2. Archivo `/etc/devfs.conf`[7] se modificó agregando enlaces simbólicos para reconocer a la unidad `/dev/cd0` (unidad creada por el uso de `camcontrol` a partir de `/de/acd0`) como `/dev/cdrom` con la línea:

   ```
   link cd0 cdrom
   ```

   Además, se permite a los usuarios tener acceso y permiso de uso de unidades CD y unidades **USB** con líneas como las siguientes:

   ```
   perm  /dev/acd0 0666  # acd* es para dispositivos CD.
   perm  /dev/da0  0666  # da* espara dispositivos USB.
   ```

3. El archivo `/etc/devfs.rules`[8] es creado para definir para la configuración de los dispositivos en el sistema.

4. El archivo `/etc/hosts`[9] es modificado dinámicamente por un *script* de instalación del sistema.

5. El archivo `/etc/make.conf`[10] se le agregaron las líneas:

   ```
   USE_GLX=yes
   CUPS_OVERWRITE_BASE=yes
   ```

6. El archivo `/etc/rc.conf`[11] es configurado principalmente desde el momento en que se instala el sistema y actualizado desde los servicios de **PC-BSD**. En éste se ingresan datos como el nombre de la máquina, configuración de ratón, configuración de las interfaces de red, además de habilitar y deshabilitar los demonios y servicios como el de **SSH**, el demonio de **USB**, **CUPS**, **Samba**, **HAL** y muchos más.

[6] El archivo `/etc/crontab` tiene el propósito de mantener las tablas de acciones de todos los usuarios para ejecutar comandos periódicamente mediante `cron`.

[7] El archivo `/etc/devfs.conf` tiene la finalidad de configurar los dispositivos cuando el sistema arranca.

[8] El archivo `/etc/devfs.rules` establece un conjunto de reglas usadas junto con `devfs.conf`.

[9] El archivo `/etc/hosts` funge como una base de datos de nombres de los hosts.

[10] El archivo `/etc/make.conf` contiene información sobre como construir el sistema cuando éste se compila desde cero.

[11] El archivo `/etc/rc.conf` carga todos los servicios que se necesitan en el sistema.

7. El archivo `/etc/rc.shutdown`[12] se le agregó la ejecución de otros *scripts* pertenecientes a **PC-BSD**. La línea agregada es la siguiente:

   ```
   #Insert other shutdown procedures here
   /PC-BSD/Scripts/UpdateHints.sh
   ```

8. El archivo `/etc/sysctl.conf`[13] se altera como se muestra a continuación:

   a) Se activó la opción de deshabilitar el `coredump` con la definición de esta variable de núcleo:

   ```
   kern.coredump=0
   ```

   b) Se necesitaba que los usuarios pudieran montar unidades como CDs. Esto se logra definiendo a "1" la siguiente variable de núcleo:

   ```
   vfs.usermount=1
   ```

   c) Por último otra modificación fue la de definir que se crearan más canales de audio definiendo lo siguiente:

   ```
   hw.snd.pcm0.vchans=4
   hw.snd.maxautovchans=4
   ```

9. El archivo `/etc/ttys` fue modificado para que inicie desde el principio el modo gráfico. La modificación se da para posibilitar la ejecutar `kdm`[14] desde el inicio del sistema. Para ello se cambio la línea 1 por la línea 2 como se ve a continuación:

   ```
   1 ttyv8 "/usr/X11R6/bin/xdm -nodaemon" xterm off secure
   2 ttyv8 "/usr/local/bin/kdm" xterm on secure
   ```

Otro archivo dentro del directorio `/etc`, cuyo contenido es interesante es `/etc/rc.local`[15]. Es por ello que se describe más a fondo en la siguiente parte del análisis.

[12] El archivo `/etc/rc.shutdown` describe que operaciones de apagado tiene que realizar el sistema.

[13] El archivo `/etc/sysctl.conf` tiene como propósito es definir el estado de las variables del *kernel* del sistema.

[14] El binario `kdm` es el gestor de sesiones que **KDE** proporciona.

[15] El archivo `/etc/rc.local` contiene comandos y *scripts* para el inicio del sistema.

3.3.3. Análisis de `/etc/rc.local`

El archivo /`etc/rc.local` en una instalación limpia de **FreeBSD** generalmente no está presente. En **PC-BSD** éste define acciones que se deben ejecutar al iniciar el sistema, tales acciones van desde el configurar al Servidor **X11**, cargar los módulos de sonido adecuados entre otros *scripts* con varios propósitos.

El codigo mostrado en el Cuadro 7, tiene la finalidad de ver si existe algún archivo de configuración `/etc/X11/XF86Config` para el Servidor X11. De no ser así, se ejecuta el *script* `/PC-BSD/cardDetect/x_config.sh`, cuya tarea consiste en crear un archivo de configuración inicial `XF86Config` con destino en `/etc/X11`.

```
# If no XF86Config file, try to create it
if [ ! -f /etc/X11/XF86Config ]; then
 echo "Creating XF86Config file..."
 /PC-BSD/cardDetect/x_config.sh
fi
```

Cuadro 7: Declaración de *script* de configuración del servidor **X11**.

Las líneas del Cuadro 8, ejecuta el *script* `sound_detect.sh` ubicado en `/PC-BSD/cardDetect`. Tal *script* tiene la finalidad de cargar, si es posible, el módulo adecuado de sonido a partir de datos arrojados por el comando `dmesg`[16] sobre la tarjeta de sonido, de no detectar la tarjeta intenta cargar todos los controladores posibles mediante la instrucción `kldload /boot/snd_driver`.

```
# Detect sound-card
/PC-BSD/cardDetect/sound_detect.sh
```

Cuadro 8: Declaración del *script* de detección del sonido del sistema.

Las líneas del Cuadro 9, indican la ejecución de uno de los servicios que **PC-BSD** ofrece como sistema de escritorio. Tal servicio se describe en forma

[16]El comando `dmesg` muestra todos los mensajes arrojados por el *kernel*.

detallada más adelante como servicio de detección de dispositivos de **PC-BSD**.

```
# Detect sound-card
/PC-BSD/scripts/startDetection.sh
```

Cuadro 9: Declaración del *script* de detección del dispositivos CD.

El segmento del Cuadro 10, habilita la emulación de procesos **Linux** cargando y montando el modulo `linprocfs` en `/sur/compat/linux/proc`. Tal módulo será necesario para los servicios como **HAL** y para la ejecución de algunas aplicaciones.

```
# Enable Linux Procfs
kldload linprocfs
mount -t linprocfs linprocfs /usr/compat/linux/proc
```

Cuadro 10: Declaración para cargar y montar los módulos del *kernel* de **Linux**.

Por último las líneas del Cuadro 11, tienen por objeto el ejecutar al *script* `CheckUpdates.sh` ubicado en `/PC-BSD/Scripts`. *Script* que checa si hay actualizaciones en línea del sistema.

```
# Start Online update check in 2 minutes
echo "Starting Online Update check"
(sleep 120; /PC-BSD/Scripts/CheckUpdates.sh SILENT STARTUP) &
```

Cuadro 11: Declaración del *script* de actualizaciones de **PC-BSD**.

En un análisis resumido, así se comporta el archivo de `/etc/rc.local` dentro del sistema. Archivo que nos muestra que gran parte de las operaciones nuevas del sistema se lanzan como una serie de *scripts* de los cuales se hablará en la siguiente parte del análisis.

3.4. Análisis de la jerarquía `/PCBSD`

Como cualquier sistema **Unix**, **PC-BSD** centra la salida de la mayoría de sus operaciones en un directorio principal encontrado en la raíz del sistema. Tal directorio contiene binarios, *scripts* y de otros tipos de archivos.

La jerarquía y una descripción simple del contenido del directorio `/PC-BSD` se muestra a continuación:

1. `bin`: contiene *scripts* y binarios usados durante la instalación, eliminación y actualización de programas, así como otras operaciones del sistema.

   Los *scripts* pertenecientes a éste directorio son los siguientes:

   a) `CrashHandler` y `CrashHandler-bin:` *script* y binario encargados del manejo de errores que se producen en la instalación de programas.

   b) `InstallLang:` *script* que llama a ejecutar al binario encargado de la instalación de idiomas que lleva el mismo nombre.

   c) `LangFindCD:` *script* que determina si un disco contiene los *paquetes* de idiomas.

   d) `LangPkgAdd:` *script* que instala un *paquete* de idioma determinado en **KDE**.

   e) `PBIdelete:` binario encargado de la operación de borrado de aplicaciones **PBI** instaladas en el sistema.

   f) `PBIUpdater` y `PBIUpdater-bin:` *script* y binario para el proceso de actualización de aplicaciones.

   g) `PCBSDUpdater` y `PCBSDUpdater:` *script* y binario que controlan el proceso de actualización del sistema.

   h) `wget:` binario para descargar de archivos desde **Internet**.

2. El directorio `cardDetect` contiene principalmente *scripts* usados para la configuración de dispositivos como la tarjeta de video del sistema, la tarjeta de sonido y algunos servicios del sistema.

   Los *scripts* podrían dividirse de la siguiente manera:

a) *Scripts* para configuración de **X11**: Los archivos pertenecientes a éste grupo son los encargados de configurar adecuadamente al servidor **X11**. Ha esta lista pertenecen lo siguientes archivos:

1) `XF86Config.default:` archivo de configuración por defecto de **X11**.
2) `XF86Config.vmware:` archivo de configuración por defecto de **X11** para máquinas virtuales.
3) `xfree_layout:` *script* encargado de configurar la disposición del teclado del sistema.
4) `x_config:` *script* encargado de generar el archivo de configuración inicial para el servidor **X11**.
5) `nvidia_card:` *script* de configuración del controlador **Nvidia** para **XFree86**.
6) `videocard_dri:` *script* usado para tarjetas de video **Nvidia**.
7) `nvidia_on y nvidia_off:` ambos archivos contienen librerías destinadas para el uso de tarjetas de video **Nvidia**.

b) *Scripts* para la configuración de la tarjeta de sonido del sistema. A éste grupo pertenecen los siguientes archivos:

1) `sound_detect:` Este *script* es el encargado de cargar el módulo adecuado para la tarjeta de sonido con la que cuenta el sistema a partir de datos generados por `dmesg`.
2) `sound_defines:` Contiene una lista de controladores para tarjetas de sonido que presentan cierto tipo de valores.
3) `verific_sunet:` *script* encargado de determinar si el controlador de sonido está cargado. Si no lo está se llama al *script* `sound_detect` para configurar la tarjeta de sonido.

3. El directorio `Computer` contiene datos de los dispositivos que han sido generados por el *script* de `refresh_fstab`.

4. Directorio `conf` contiene los archivos que controlan es el funcionamiento del sistema **PC-BSD**. Lo archivos presentes en este directorio son los siguientes:

a) Archivo `PCBSDv1.3:` Es el archivo de configuración del *kernel* usado por el sistema.

b) Archivo `PCBSDv1-SMP`: Archivo de configuración del *kernel* usado para máquinas con varios procesadores.

c) Los archivos `.daily`, `.monthly` y `.weekly`. Archivos usados para las funciones periódicas del sistema.

5. El directorio `docs` contiene la guía oficial del sistema en formato *html*.

6. El directorio `files` contiene al archivo `AddIcon.png`, el cual es usado por los binarios del sistema.

7. El directorio `images` contiene imágenes usadas por los binarios de **PC-BSD**.

8. El directorio `kernels` contiene los archivos de configuración para los *kernels* del sistema. Aquí se encuentran: `kern-1.3-standard` para máquinas con un solo procesador y `kern-1.3-SMP` para máquinas con múltiples procesadores.

9. El directorio `LANGS` contiene archivos de traducción de varios idiomas usados en lo procesos de instalación del sistema.

10. El directorio `Patches` contiene directorios vacíos que simbolizan los parches por los que ha pasado el sistema desde la versión *0.8* hasta la versión *1.11a*.

11. El directorio `Scripts` contiene la mayoría de los *scripts* importantes del sistema como lo son el servicio de detección de dispositivos. Los *scripts* que conforman este directorio son los siguientes:

    a) `startDetection.sh`: *script* que se encarga de encontrar todos los dispositivos CD de sistema para empezar el proceso de detección de medios a través del *script* `detectMedium.sh`. En esencia es el archivo principal para el servicio de detección de medios.

    b) `detectMedium.sh`: *script* cuyo propósito es determinar cuando un disco CD ha sido introducido en algún dispositivo CD.

    c) `fsReplace.sh`: *script* que se encarga de introducir y remover las entradas de dispositivos CD a `/etc/fstab` produciendo y eliminando accesos a los dispositivos CD para cada usuario.

    d) `adduser.sh`: *script* para agregar nuevos usuarios al sistema.

e) `CheckPBIUpdates.sh`: *script* encargado de examinar actualizaciones de algunas de las aplicaciones **PBI** en el sistema.

f) `CheckUpdates.sh`: *script* que se encarga de examinar si hay actualizaciones disponibles para el sistema.

g) `installPatch.sh`: *script* encargado de instalar algún parche del sistema.

h) `registerPatch.sh`: *script* que registra el parches aplicados al sistema.

i) `runpbi.sh`: *script* encargado de empezar el proceso de instalación de una aplicación **PBI**.

j) `refresh_fstab.sh`: *script* que examina a `/dev` para ver si hay nuevos dispositivos y agregarlos a `/etc/fstab` y a `/mnt` si es que todavía no están. Si en `/etc/fstab` hay dispositivos desconocidos serán removidos del archivo.

k) `AutorunCD.sh`: *script* encargado de examinar si una unidad CD contiene una archivo denominado `autorun.pbi`, si tal archivo existe se ejecuta éste.

l) `diskIcons.sh`: *script* encargado de crear accesos a los nuevos dispositivos del sistema en los escritorios de los usuarios.

m) `create_drives.sh`: *script* cuyo propósito es crear entradas a los escritorios con base a la información de `/etc/fstab`.

Además de los archivos anteriores existe dentro de éste mismo directorio otro llamado `System`, el cual contiene *scripts* usados por el sistema en algunas operaciones importantes. Los archivos dentro de éste directorio son:

a) `InstallKernel.sh`: *script* con la tarea de instalar un nuevo *kernel* al sistema.

b) `GenDiagSheet.sh`: *script* que genera un archivo sobre el estado del sistema.

c) `Portsnap.sh`: *script* encargado de actualizar del sistema de *puertos* del sistema.

d) `Changes.sh`: *script* usado por la actualización de los *puertos* de FreeBSD.

12. Directorio `Services:` contiene directorios que representan los servicios que **PC-BSD** maneja, en donde cada directorio contiene una serie de *scripts* que dominan alguna operación especifica del servicio. Tales propiedades son representadas de la siguiente manera:

    a) `disable:` *script* que encargado de deshabilitar el servicio en cuestión.

    b) `enable:` *script* que cuyo propósito es habilitar al servicio dentro del sistema.

    c) `isEnabled:` *script* que examina si el servicio en cuestión esta habilitado.

    d) `isRunning:` *script* determina si un el servicio en cuestión esta ejecutándose en el sistema.

    e) `restart:` *script* que se encarga de re inicializar al servicio en cuestión.

    f) `start:` *script* que se encarga de inicializar al servicio en cuestión.

    g) `stop:` *script* que se encarga de detener al servicio den cuestión.

    h) `service:` archivo extra especificando el nombre del servicio en cuestión.

    Definidas ya las propiedades de los servicios, se muestran a continuación el nombre de cada uno de ellos:

    a) `cdautorun:` Servicio encargado de la detección de dispositivos CD.

    b) `cups:` Servicio de impresión por defecto de **PC-BSD**.

    c) `denyhost:` Servicio para bloquear lo ataque realizados vía **SSH**.

    d) `fstab:` Servicio de recrear a **/etc/fstab** en cada inicio del sistema.

    e) `hal:` Servicio del sistema para la interacción con el hardware.

    f) `pf:` Servicio de *Firewall* del sistema.

    g) `pfrules:` Servicio para generar las reglas de ***Package Filter (PF)*** en cada inicio del sistema.

    h) `ssh:` Servicio de acceso encriptado **SSH** del sistema.

13. El directorio `sound` contiene una versión del proceso de detección del módulo de sonido del sistema.

14. Por último, el directorio `splash-screens` contiene todos los protectores de pantalla del sistema en todos los idiomas los cuales son usados durante el proceso de inicio del sistema.

3.5. Servicios de PC-BSD

3.5.1. Servicio de detección de dispositivos mediante HAL

La auto detección de dispositivos en **PC-BSD** en general es proporcionado por **HAL (Hardware Abstraction Layer)**. Tal *paquete* tiene como objetivo proveer información sobre los dispositivos del sistema. Para que otros programas (tales como aplicaciones de escritorio) puedan localizar y utilizar los dispositivos de hardware.

Los dispositivos que **HAL** soporta son aquellos que van desde unidades de disco compactos, discos de almacenamiento masivo tales como memorias **USB**, reproductores de audio, cámaras. También es capaz de proporcionar acceso a nuevos discos duros que hayan sido introducidos en el sistema.

3.5.2. Servicio de detección de discos CD/DVD de PC-BSD

El servicio de autodetección de discos CD es proporcionado por el sistema desde la versión *0.8* y poca de su estructura e idea han cambiado hasta la versión *1.3*. La idea principal de tal servicio es dar a los usuarios un acceso a los dispositivos CD en el momento en que éstos fueran introducidos en alguna unidad del sistema. Teniendo el usuario la opción poder montar la unidad y desmontarla en el instante que éste quisiera. En el momento que el disco CD fuera sacado de la unidad CD, el acceso al dispositivo es borrado automáticamente.

La ubicación de tal servicio se encuentra en `/PCBSD/Scripts/` y lo conforman tres *scripts* los cuales son `startdDetection.sh`, `detectMedium.sh` y `fsReplace.sh` los cuales son llamados desde `/etc/rc.local`. A continuación se describe cada de las operaciones de los *scripts* con las líneas más importantes de su código.

Descripción de `startDetection.sh`

La idea original de éste *script*, se basa en detectar si el sistema cuenta con unidades CD con nombre `cd0`,`cd1` y así sucesivamente. Para hacer esto se hace uso del comando `camcontrol` como se ve a continuación.

```
camcontrol devlist | grep cd0 > /dev/null
```

Si alguno de estos dispositivos existen (en este caso cd0), se tratará de encontrar algún identificador asignado dinámicamente al dispositivo.

```
PASS=‘camcontrol devlist | grep "cd0" |
cut -f 2 -d "(" | cut -f 1 -d "," |
cut -f 1 -d ")"‘
```

Al encontrar el identificador asignado al dispositivo, se crea un directorio en donde va a radicar el contenido de dispositivo CD cuando éste sea montado. Hecho lo anterior, se manda a ejecutar al *script* `detectMedium.sh` como un proceso en segundo plano con los datos de entrada como los son el identificador encontrado y el nombre del dispositivo que se va a monitorear.

```
(nice /PCBSD/Scripts/detectMedium.sh $PASS cd0) &
```

Si no existe tal dispositivo, no se hace nada (puesto que este tipo de unidades no se pueden agregar dinámicamente al sistema cuando éste se encuentra en ejecución).

El procedimiento es en cierto punto genérico y puede ser usado para n unidades CD que se quieran examinar. PC-BSD por defecto fue diseñado para poder soportar hasta 4 dispositivos de éste tipo.

Descripción de `detectMedium.sh`

Su objetivo es determinar el punto de montaje con el nombre del dispositivo que se pasa como parámetro. Hecho lo anterior, se entra en un ciclo infinito de verificación en donde se examina si el servicio de autodetección de dispositivos aun está habilitado, de no ser así, el ciclo infinito y el servicio termina su ejecución.

```
if [ ! -e "/PCBSD/conf/enable\_cd\_autorun" ]; then
        exit
fi
```

En el caso de estar habilitado el servicio (la habilitación o no habilitación del servicio se controla con la existencia del archivo `enable_cd_autorun` localizado `/PCBSD/conf`), se examina si el dispositivo está listo o no, es decir; estar listo significa que dentro del dispositivo hay un medio CD presente listo para ser usado.

```
test="‘camcontrol tur ${1} -v >&1 2>&1‘"
echo $test | grep "Unit is ready" > /dev/null 2>/dev/null
```

Si esta listo el dispositivo, y si no hay algún archivo de registro de este en el sistema, entonces se ejecuta al *script* `fsReplace.sh` para notificar a los escritorios de los usuarios del sistema sobre el suceso.

```
/PCBSD/Scripts/fsReplace.sh $MNTPNT
```

Si no está listo el dispositivo, se checa si existe un archivo de registro que indica que éste dispositivo estuvo listo en algún momento y que se notificó a los usuarios, de existir tal registro, se procede de buscar en todos los escritorios de los usuarios algún acceso a dicho dispositivo para su eliminación puesto que ya no es vigente en el sistema.

```
for i in ‘ls‘; do
        if [ -e "/home/${i}/Desktop/${2}" ]; then
                rm /home/${i}/Desktop/${2}
        fi
done
```

El procedimiento anterior se repite n veces hasta el que sistema se apague o en el momento que tal servicio ya no esté habilitado.

Descripción de `fsReplace.sh`

La actualización de `/etc/fstab` recae en el trabajo de este *script*, este checa si en el archivo `/etc/fstab` está el dispositivo que ha sido enviado como parámetro de entrada. Si no existe el contenido de `/etc/fstab` pasa de la misma manera.

```
cat /etc/fstab | grep -v ${1} > /tmp/fstab.tmp
mv /etc/fstab /etc/fstab.bak
mv /tmp/fstab.tmp /etc/fstab
```

Hecho lo anterior se procede a ingresar una línea de configuración del dispositivo CD a `/etc/fstab` alterando su contenido.

```
echo "/dev/${DEV} /mnt/${1} auto ro,noauto 0 0" >> /etc/fstab
```

Por último se examina que todos los usuarios tengan un acceso al dispositivo creando en cada escritorio del usuario un acceso para que ellos puedan montar y desmontar a voluntad el medio CD presente.

El servicio presentado anteriormente es bueno, pero lamentablemente padece de errores. Uno de ellos es el acceso y modificación al archivo `/etc/fstab` de manera constante. En casos graves, tal actualización puede terminar borrando el contenido completo de `/etc/fstab`. Otro error se da al no ser capaz de distinguir entre un CD de datos y un CD de audio. Tal punto es clave puesto que los discos de audio no tienen un sistema de archivos, son simplemente son pistas por lo cual al tratar de montarlos se producen errores. Estos dos errores se sufrían en versiones anteriores a *1.3* puesto que **HAL** no era parte del sistema.

3.5.3. Servicio de impresión CUPS

Este servicio de impresión del sistema es proporcionado en esencia por el *paquete* ***CUPS (Common Unix Printing System)*** el cual provee una capa de impresión portable para sistemas operativos basados en **Unix**. Este usa el ***Protocolo de Impresión de Internet (IPP)*** como base para manejar trabajos y colas de impresión, cuenta con el ***Demonio de Impresión en Línea (LPD)***, al ***Servidor de Mensajes de Bloque (SMB)*** y los protocolos de **AppSocket** que son soportados con funcionalidad reducida.

La aplicación **CUPS** se encuentra ya precompilado en los *paquetes* de **FreeBSD**. Generalmente es usado con otros *paquetes* adicionales para obtener un mayor aprovechamiento del sistema de impresión entre los cuales destacan **Samba**.

Con lo referente a la configuración del archivo `/etc/printcap` de las impresoras, éste es configurado por **CUPS**.

3.5.4. Servicio para compartir directorios y archivos Samba

El Servicio para compartir archivos y directorios dentro del sistema es mediante la utilización de la suite ***Samba (SMB and CIFS client and server for Unix)***. **Samba** es un conjunto de programas que implementan un servidor de archivos **Windows** y protocolos para compartir impresoras **(SMB/CIFS)**. Además de permitir a clientes **Windows** usar un espacio en el sistema de archivos e impresoras de los sistemas **BSD** como componentes locales.

La aplicación **Samba** y sus componentes ya están precompilados para **FreeBSD** por lo que solo se necesita del sistema base y algunas librerías para disfrutar de este servicio.

El archivo de configuración `/etc/smb.conf` (archivo ubicado en `/etc` porque se hizo un *link* simbólico de `/etc/samba/smb.conf` a `/etc/smb.conf`) tiene definido por defecto como grupo de trabajo a **LAN** y como nombre del servidor local a **PC-BSD**.

3.6. Instalación de aplicaciones mediante *paquetes* PBI

Una de las características de PC-BSD es el desarrollo de una nueva herramienta gráfica para la instalación de software basado en *paquetes .pbi*. Los *paquetes* PBI son archivos específicos del sistema. Internamente un *script* se ejecutará con el archivo para ver si es un *script* de *consola* o un binario y así tomar las acciones apropiadas para su funcionamiento.

Los archivos **PBI** son creados mediante un programa desarrollado por el grupo **PC-BSD** de nombre ***PBC (PCBSD Creator) o PBI Creator*** como se conoce comúnmente. La aplicación **PBC** permite crear los *paquetes* de fácil instalación con la extensión *.pbi*. En donde se configuran opciones para su funcionamiento tales como los directorios de las librerías necesarias, iconos para el menú de **KDE** e iconos del escritorio.

Hasta cierto punto ésta idea envuelve conceptos al **Microsoft Windows** puesto que se cuenta con un ***Asistente (Wizard en Ingles)*** que instala tales *paquetes*. La única acción que realiza es copiar el contenido que viene dentro del archivo con extensión **PBI** a un directorio destino ya determinado.

Capítulo 4

Implementación y Desarrollo

4.1. Aplicaciones dentro de X-BSD

Una de las principales características de **X-BSD** es sin duda la opción de poder tener aplicaciones enteras en sólo un directorio. A tales aplicaciones se les nombra como *Application Directory*[1]. Un directorio de éste tipo contiene archivos de ayuda del programa, imagenes, binarios y posiblemente otras cosas. La aplicación dentro del directorio no necesita que el sistema tenga instalado aquellos *paquetes* o librerías requeridas para su ejecución.

4.1.1. El concepto de *directorio-app*

Un *"directorio-app"* no necesita de un instalador, es por ello que uno puede moverlo a donde uno quiera. Uno puede desinstalar una aplicación simplemente eliminando el directorio. Si se desea correr la aplicación simplemente se debe hacer click en el directorio.

El concepto de *directorio-app* no es nuevo, ellos son usados por **RISC OS**, **NextStep**, **MAC OS X** y en programas como el manejador de archivos **ROX**[2]. Por ejemplo, dentro de **MAC OS X** su filosofía empieza desde el empaquetado de la aplicación. Las aplicaciones en **MAC OS X** se pueden descargar fácilmente desde **Internet** como archivos con extensión **.dmg* ***(Disk iMaGe)***. Para poder visualizar el contenido de éste tipo de archivos

[1] De aqui en adelante nos referiremos a un *Application Directory* como un *directorio-app*.
[2] http://roscidus.com/desktop

simplemente se necesita abrir al archivo y el sistema creará un acceso a una *unidad de virtual* con el mismo nombre del archivo **.dmg*. La ubicación en donde se creará el acceso será en el directorio donde se encuentre el archivo **.dmg*.

Dentro de **MAC OS X** una *unidad virtual* puede ser desmontada simplemente enviando tal acceso a la papelera (sigue la misma filosofía de las memorias USB). El contenido de la *unidad virtual* es un directorio con extensión **.app* el cuál contiene dentro una aplicación para el sistema.

La razón por la cual **MAC OS X** no maneja tal directorio como los demás, es porque el sistema sabe que no se trata de un directorio común sino de una aplicación. Tal distinción se da por el hecho de existir un archivo **.xml* dentro del directorio que define sus propiedades. Una de ellas es la opción de poder visualizar el contenido del *directorio-app*.

La idea explicada anteriormente fue retomada casi en su totalidad por su *simplicidad* para adoptarla dentro de **X-BSD** como una mejor opción a los **PBI** de **PC-BSD**.

El motivo de no elegir la idea de las aplicaciones **PBI** de **PC-BSD** fue el hecho de que varias de estas aplicaciones necesitan de enlaces simbólicos. Tales enlaces son generados por un *script* en el momento que se instala la aplicación dentro del sistema, con el fin de que tal aplicación pueda funcionar de manera correcta. Llegado el momento cuando el usuario desinstala la aplicación. Un *script* es ejecutado para realizar esta limpieza de enlaces simbólicos que había hecho la aplicación en su fase de instalación.

La idea en principio es buena, puesto que el propósito de **PC-BSD** es ofrecer un instalador que permita a los usuarios comunes el poder instalar aplicaciones simplemente haciendo un click en el archivo con extensión **.pbi*. Las principales desventajas sobre las aplicaciones **PBI** se presentan en el momento de instalar aplicaciones mediante el comando `pkg_add`. Tal comando examina si los archivos que se debe de extraer del *paquete* a instalar no existen ya en el sistema, de no existir tales archivos, ellos son extraídos de manera correcta terminando así la instalación de *paquete*. El problema se presenta cuando éste encuentra enlaces simbólicos de aplicaciones instaladas mediante los *paquetes* **PBI**. Enlaces que poseen el mismo nombre de alguno

de los archivos a extraer. Esto genera que no sea posible instalar el *paquete* hasta que se de la eliminación de tales archivos que ya están presentes dentro del sistema.

Otro problema se presenta cuando el sistema es actualizado. Tal proceso elimina varios archivos situados en `/usr/local`. Algunos de esos archivos, son los enlaces simbólicos los cuales fueron creados por algunas aplicaciones **PBI** durante su fase de instalación. Es por ello que, terminada la actualización del sistema, la aplicación **PBI** queda imposibilitada de ejecutarse de nuevo.

Declaradas las causas de elegir la idea de los *directorios-app* de **MAC OS X**, procedamos con definir esta filosofía de aplicaciones dentro de nuestro sistema.

1. Se define un nuevo tipo de archivo dentro dentro de **KDE** con extensión **.dmg*. Este tipo de archivo será una partición del sistema, es decir, con sistema de archivos **FFS**. Al momento de intentar abrir éste tipo de archivos, se creará una *unidad virtual*.

2. Una *unidad virtual*, en realidad es un nodo virtual usado por el comando `vnconfig`. Tal nodo será montado permitiendo su acceso gracias a un enlace o *link* (archivo con extensión **.desktop*) creado en el mismo directorio del archivo **.dmg*. Si uno desea desmontar la *unidad virtual* simplemente se necesita borrar al archivo **.desktop* creado.

3. Dentro del sistema, un directorio puede ser también una aplicación con la característica de poseer una extensión *.app* y una jerarquía que lo distingue de los demás. Este tipo de directorios, estarán contenidos dentro los archivos **.dmg* para el sistema.

   La jerarquía de un *directorio-app* se describe a continuación:

   a) `.directory`: éste archivo de configuración es el encargado de asignar un icono referente a la aplicación para hacer distinción de los directorios normales.

   b) `Icon`: directorio que contiene un archivo con nombre `icon` y con extensión *jpeg, png o xpm*. El archivo imagen tiene por objetivo ser el icono de la aplicación.

c) `Info`: éste directorio es donde se encuentran aquellos archivos que muestran información asociada a la descripción y contenido de la aplicación.

d) `lib`: el contenido de tal directorio son todos aquellos archivos y librerías que son indispensables para poder ejecutar la aplicación de una manera local.

e) `bin`: dentro de tal directorio se encuentran los archivos binarios de la aplicación.

f) `app-launcher.sh`: *Script* cuyo objetivo es hacer los preparativos necesarios para poder ejecutar la aplicación. Dentro de éste script se distinguen tres fases las cuales son definidas más adelante.

Respecto a otros directorios no mencionados que se encuentren dentro de la jerarquía de la aplicación. Ellos pueden incluirse si son necesarios durante la ejecución del binario de la aplicación como se muestra en el Cuadro 12, en donde se lista el contenido del *directorio-app* de la aplicación `mozilla-firefox` mostrando todos los archivos y directorios de su jerarquía.

```
-rwxr-xr-x   1 root  wheel    37 Jan  9 09:12 .directory
drwxr-xr-x   2 root  wheel   512 Jan  9 09:09 Icon
drwxr-xr-x   3 root  wheel   512 Jan  9 09:09 Info
-rwxr-xr-x   1 root  wheel  1244 Jan  9 09:12 app-launcher.sh
drwxr-xr-x   2 root  wheel   512 Jan  9 09:06 bin
drwxr-xr-x   4 root  wheel  1024 Jan  9 09:06 lib
drwxr-xr-x  14 root  wheel  1024 Jan  9 09:09 mozilla-firefox
```

Cuadro 12: Ejemplo de la jerarquía de un *directorio-app*.

4.1.2. El *script* `app-laucher.sh` *(Application Launcher)*

El *script* `app-laucher.sh` es por si mismo el más importante dentro de la jerarquía de un *directorio-app*. Su contenido es quien define las operaciones a realizar antes de la ejecución de la aplicación.

La estructura de éste archivo consta de tres fases la cuales están bien diferenciadas una de otra como se muestra en el Cuadro 13. La primera fase es la *"Fase de declaración de variables globales"*, en donde por definición deben de existir las variables `PATH_APP` y `LD_LIBRARY_PATH`. La variable `PATH_APP` guardará la ruta del *directorio-app*. La variable `LD_LIBRARY_PATH` define la ruta en donde están ubicadas las librerías que debe usar la aplicación.

```
#!/bin/sh
# Declaración de fases de Application Launcher

## Fase de declaración de variables globales...
PATH_APP=$1; export PATH_APP
LD_LIBRARY_PATH="${PATH_APP}/lib" ; export LD_LIBRARY_PATH
...
...

##  Fase de configuración de archivos de paquetes...
...
...

## Fase de ejecución de la aplicación...
${PATH_APP}/bin/binario-de-la-aplicación
```

Cuadro 13: Estructura del script app-laucher.sh

En ésta fase también pueden incluirse otras variables que se necesiten para lograr que la aplicación funcione de manera correcta. La forma de declaración de variables se muestra en el Cuadro 14. Un ejemplo de otra variable es `PANGO_RC_FILE`, la cual es solicitada cuando se trabaja con el *paquete* `pango`.

```
VAR\_EJEMPLO="\${PATH\_APP}/otro-valor"; export VAR\_EJEMPLO\\
```

Cuadro 14: Declaración de una variable dentro de app-laucher.sh

La segunda fase es la *"Fase de configuración de archivos de paquetes"*, en donde se crean o configuran archivos que deben de existir por ser solicitados

por algunas librerías. En la mayoría de los casos sólo es el cambio de algunas rutas sobre algunos archivos ubicados dentro del directorio de librerías `lib`. Esta fase puede existir o no dependiendo de las necesidades de la aplicación.

La última fase es la *"Fase de ejecución de la aplicación"*, en donde se define el nombre del binario a ejecutar. Su estructura siempre respeta un patrón en donde sólo cambia el binario a ejecutarse.

4.1.3. Proceso de *alteración de librerías al nivel binario*

El *proceso de alteración de librerías al nivel binario*[3] es un *hackeo*[4] que intenta cambiar el comportamiento interno de un archivo binario (librería) utilizando la idea del intercambio de caracter por caracter. En éste proceso se cambia una serie de caracteres por otra serie del mismo tamaño sin alterar el tamaño del archivo binario. Con ello se evita la provocación de una incongruencia interna del binario (para el caso de librerías, la producción de una librería rota).

El proceso se originó con el fin de hacer que varias librerías situadas en un mismo directorio, dependan de ellas mismas y no busquen otros binarios en lugares distintos a su ubicación. Con ello se la opción de poder tener un concepto similar a las llamadas *aplicaciones portables*[5].

El *proceso de alteración al nivel binario* respeta los siguientes pasos:

1. El primer paso para poder hacer una alteración de alguna librería es identificar aquellas que se deben de modificar, muchas de las veces

[3]En algunos experimentos sobre **OpenBSD**, las modificaciones hechas a los archivos binarios fueron exitosas. No sabemos si tales modificaciones se comporten de la misma manera en otros sistemas.

[4]Tal *hackeo* puede evitarse compilando dichas aplicaciones, en donde se define que todas las librerías requeridas en tiempo de ejecución siempre serán locales.

[5]Una *aplicación portable* es una aplicación informática que puede ser utilizada en cualquier ordenador que posea al sistema operativo para la que fue programada, ello implica que no se requiere la instalación de bibliotecas adicionales en el sistema para su funcionamiento.

estas librerías son aquellas que piden la existencia de algún archivo en un lugar específico. Tal ruta es la cadena a reemplazar.

2. Al tener en mente la cadena a cambiar dentro de las librerías, se procede a buscar cuales de todas aquellas librerías que utiliza la aplicación son las que contienen tal cadena como se ve en el Cuadro 15, en donde la librería `libcairo.so.5.0` es requerida por otras tres librerías dentro de una aplicación.

```
# grep libcairo.so.5.0 *
Binary file libgdk-x11-2.0.so.802.1 matches
Binary file libgtk-x11-2.0.so.802.1 matches
Binary file libpangocairo-1.0.so.1200.3 matches
```

Cuadro 15: Búsqueda de la cadena `libcairo.so.5.0` con ayuda del comando `grep`.

3. El tercer paso es buscar dentro de la librería (con ayuda de un editor binario) a la cadena en donde se necesita modificar su significado. Tal cadena generalmente indica la ruta de algún archivo binario.

   Por ejemplo `/usr/local/lib/libpangoft2-1.0.so.1200.3` puede ser cambiada por `lib-xxxxxxxxxx-libpangoft2-1.0.so.1200.3`. En donde el valor de *"x"* puede ser cualquier caracter que uno desee, siempre y cuando se respete la convención sobre la cadena resultante. La cual siguiere que la cadena de reemplazo debe comenzar con los caracteres `lib` y terminar con el nombre de la librería a alterar.

   Terminada la modificación, se actualiza la librería siempre y cuando no se haya modificado el tamaño original del binario. Si durante el proceso se modificó tal tamaño, no se actualiza nada y procede a comenzar de nuevo el proceso. De hacerlo bien se actualiza la librería.

4. Para finalizar la alteración de la librería solo falta crear un enlace de la librería local a librería que hemos definido, todo esto si tal enlace aún no existe. En el Cuadro 16, se puede ver la creación del enlace simbólico a `lib-xxxxxxxxxx-libpangoft2-1.0.so.1200.3` desde de la librería `libpangoft2-1.0.so.1200.3` con ayuda del comando `ln`.

```
# ln -s libpangoft2-1.0.so.1200.3
  lib-xxxxxxxxxx-libpangoft2-1.0.so.1200.3
```

Cuadro 16: Ejemplo de la creación del enlace intermedio entre librerías.

4.2. El sistema de aplicaciones app

Dentro de **X-BSD**, el manejo de los *directorios-app* está a cargo del *sistema de aplicaciones app*. El sistema tiene la tarea de realizar todas las operaciones necesarias para igualar un comportamiento similar a **MAC OS X**. Su implimentación está dentro del *script* `application-directory.sh` situado dentro de `/X-BSD/Bin/`[6].

4.2.1. El *script* `application-directory.sh`

El *script* `application-directory.sh` puede verse como crítico en el sistema. Su principal objetivo el de proveer al entorno de escritorio **KDE** la opción de poder trabajar con los llamados *directorios-app*.

Su funcionamiento es en cierto sentido muy básico, pero a la vez es complejo por el uso de instrucciones del sistema como lo es ***DCOP (Desktop COmmunication Protocol)***[7].

El uso de **DCOP** añade nuevas capacidades muy extensas sin el requerimiento de escribir aplicaciones completamente nuevas, como quiza fuese en otro caso. Las aplicaciones **KDE** y las librerías del mismo entorno de escritorio hacen un uso intensivo de este sistema y la mayoría de las aplicaciones pueden ser controladas mediante *scripts* a través del mismo protocolo.

En sistemas **KDE** modernos, cada aplicación tiene un conjunto básico de interfaces **DCOP**, incluso si el programador de la aplicación no lo ha codificado de manera explícita. Por ejemplo, cada aplicación admite automática-

[6]En caso de no instalar el sistema, puede consultar el contenido de todos *scripts* y archivos declarados desde éste punto en el disco de **X-BSD** en el directorio `Archivos-Tesis/X-BSD/`.

[7]El sistema ***DCOP (Desktop COmmunication Protocol)*** permite la comunicación ligero entre procesos y componentes de software. El principal fin de éste sistema es permitir la interoperación de aplicaciones y compartir tareas complejas entre éstas.

mente la orden *quit*, que al ser llamada cierra la aplicación.

La herramienta de consola llamada `dcop` puede usarse para comunicación con las aplicaciones desde línea de comandos. Una aplicación con interfaz gráfica con las mismas funciones de `dcop` es `kdcop`, el cuál permite explorar las interfaces **DCOP** desde el entorno de **KDE**.

Volviendo a nuestro *script*, su el uso de **DCOP** es para manipular el comportamiento de los procesos de `konqueror`. Teniendo el control de `konqueror`, es posible ejecutar acciones que simulen en cierta manera el funcionamiento de reconocimiento de *"directorios-app"* del programa `finder` de **MAC OS X**.

El funcionamiento de tal *script* se define en los siguientes puntos:

El primer paso es distinguir el tipo de directorio que se está intentando abrir, puesto que un directorio puede ser un directorio común o un *directorio-app*. El reconocimiento se da mediante la búsqueda de la jerarquía de los *directorios-app*, es decir, tal directorio deben contener un *script* llamado `app-launcher.sh`, al directorio de librerías `lib`, entre otros más que conforman tal jerarquía. En caso de tratarse de un *directorio-app*, se ejecuta al *script* `app-launcher.sh` para darse así el lanzamiento de la aplicación.

En caso contrario, se buscan todos los procesos de `konqueror` con ayuda del comando `dcop` para determinar en cual de todos ellos se está solicitando la apertura del directorio. En caso de no existir ningún proceso de `konqueror` corriendo, se lanza a ejecución a un nuevo proceso de `konqueror` con la opción *Open* y como argumento la ruta del directorio que se desea abrir.

4.2.2. El manejo de archivos **.dmg*

Dentro del sistema, el manejo de archivos **.dmg* es un factor importante. Los *scripts* encargados de éstas operaciones tienen la tarea de buscar nodos virtuales en el sistema y asignarlos dinámicamente cuando sean requeridos. Son también reponsables de no permitir que los pocos nodos sean mal usados, es decir, verifican que cada nodo sea asignado correctamente y sea siempre usado.

El manejo de éste tipo de archivos está basado en dos *scripts*: `dmg.sh`

cuya finalidad es asignar nodos y `dmg_agent.sh` cuyo propósito es vigilar al nodo hasta el momento en que ya no es usado.

El *script* `dmg.sh`

El *script* `dmg.sh` determina todos los datos que se necesita para poder crear asignar un nodo virtual a un archivo **.dmg*. Tales datos son la ruta en donde se encuentre el archivo **.dmg* que se quiera abrir, el nombre de la aplicación, entre otros más. Una vez que se determinaron todos lo datos, se determina si tal archivo no esta abierto y tiene un nodo ya asigando. Si esta abierto, se le asigna el mismo nodo. En caso de no estar abierto, se procede montar el contenido el del nodo virtual, se crea un archivo de registro para el archivo **.dmg* y se crea un enlace (un archivo **.desktop*) a tal unidad quien será monitoreada por el *script* `dmg_agent.sh`.

El *script* `dmg_agent.sh`

El objetivo de éste *script* es actuar como un inspector de la existencia del archivo con extension **.desktop*, el cual fue creado en el momento de abrir al archivo **.dmg*. Su trabajo se divide en dos etapas: la primera es verificar en ciertos lapsos de tiempo, si existe el enlace a la unidad virtual. Si es que existe, el proceso continuará verificando hasta que se de la eliminación del archivo. En la segunda etapa, cuando no existe tal enlace, éste *script* remueve el registro del archivo **.dmg*, desmonta la unidad virtual y finalmente libera al nodo usado.

4.3. Servicios de detección de dispositivos

El servicio para la detección de dispositivos dentro del sistema se implementa desde cero. El motivo de no usar **HAL** sobre **OpenBSD** es por el hecho que de tal *puerto* no ha sido pasado al *árbol de puertos*.

Nuestro servicio de detección se crea por la necesidad de dar una opción facíl de poder visualizar el contenido de dispositivos como lo son los discos CD, discos DVD y las memorias **USB**.

4.3.1. Servicio de detección de disco CD

La idea básica de éste servicio consiste en las siguientes ideas: primero es tratar de detectar en momento en el cual un disco CD es insertado en algún dispositivo CD. Al ser detectado el medio, se tratará de dar acceso para los usuarios quienes podrán desechar al CD o DVD cuando el se crea conveniente.

Para lograr todo lo dicho anteriormente, éste servicio requirió su desarrollo en tres pasos (cada uno manejado por un *script*), además de implementar un programa en **C** llamado `cddevctl`.

Script `ServiceDetectionCD.sh`

Es el *script* principal dentro del servicio de detección de medios CD. Su finalidad es buscar todos los dispositivos de éste tipo en sistema. Si se encuentra uno, éste le da acceso de lectura a los usuarios, y manda a ejecutar a otro *script* llamado `DetectCD.sh`. El segundo *script* se encargará en adelante del manejo del dispositivo. Tal proceso es genérico y puede repetirse para n dispositivos CD con los que cuente el sistema.

En realidad, la verdadera labor del *script* `ServiceDetectionCD.sh`, es la de actuar como un explorador de dispositivos. Por defecto, éste explorador soporta o tiene asignado buscar sólo los primeros dos dispositivos del sistema. De no existir ningún dispositivo, los *scripts* siguientes nunca llegarán a ejecutarse.

El binario `cddevctl` de X-BSD

Uno de los puntos clave del servicio de detección de CD, consiste en no contar con una manera de poder de determinar cuando un dispositivo CD contiene algún medio dentro de él. En **FreeBSD** ésto se solucionó a partir del servicio proporcionado por `camcontrol`. Pero en **OpenBSD** tal comando no tiene alguna variante. Es por ello, que se tuvo que desarrollar una manera para poder saber éste tipo de información.

La principal tarea del binario `cddevctl` es reemplazar la operación del comando `camcontrol tur` de **FreeBSD** que determina si un dispositivo CD está listo para ser leído. Un dispositivo está listo si existe un medio CD den-

tro del dispositivo.

El binario, centra la mayoría de su contenido en el uso de la función `opendev` de la biblioteca `util.h`. Internamente se usan los valores de regreso en la bandera `errno` (principalmente los valores `EBUSY` y `EIO`) cuando `opendev` es ejecutado. La manera de decir si existe algún medio CD o DVD en un dispositivo determinado se da mediante la impresión de un mensaje de confirmación. Tal mensaje puede ser *"Si hay CD!!"* o *"No hay CD!!!"*.

Otro aspecto que se relaciona con éste binario es la modificación hecha al *kernel* de **OpenBSD**. La modificación más importante es la eliminación de algunos mensajes *kernel* producidos por la función `opendev` cuando se intenta acceder a un dispositivo CD sin nugún disco CD.

El archivo modificado fue `/usr/src/sys/scsi/scsi_base.c`. Los cambios fueron hechos dentro de la implementación de la función:

```
void scsi_print_sense(struct scsi_xfer *xs)
```

La primera modificación se presenta al comentar la líneas:

```
sc_print_addr(xs->sc_link);

/*XXX For error 0x71, current opcode is not the relevant one*/
printf("%sCheck Condition (error %#x) on opcode 0x%x\n",
(serr == SSD_ERRCODE_DEFERRED) ? "DEFERRED " : "", serr,
xs->cmd->opcode);
```

El resultado de la primera modificación se ve en el Cuadro 17. La segunda modificación es comentar las líneas:

```
printf("    SENSE KEY: %s\n", scsi_decode_sense(sense,
DECODE_SENSE_KEY));
```

El resultado de la segunda modificación se ve en el Cuadro 18. Por último la tercera modificación se da al comentar las líneas:

```
 if (strlen(sbs) > 0)
printf("     ASC/ASCQ: %s\n", sbs);
```

El resultado de la modificación se ve el Cuadro 19. Hechas las modificaciones, se compila todo el sistema para obtener el nuevo *kernel*.

```
....
char                                    *sbs;

/*sc_print_addr(xs->sc_link);*/

/*XXX For error 0x71, current opcode is not the relevant one.*/
/*printf("%sCheck Condition (error %#x) on opcode 0x%x\n",
(serr == SSD_ERRCODE_DEFERRED) ? "DEFERRED " : "", serr,
xs->cmd->opcode);*/

if (serr != SSD_ERRCODE_CURRENT && serr != ...
```

Cuadro 17: Primera parte del código comentado de /usr/src/sys/scsi/scsi_base.c.

```
...
return;
}

/*printf("    SENSE KEY: %s\n", scsi_decode_sense(sense,
DECODE_SENSE_KEY));*/

if (sense->flags & (SSD_FILEMARK | SSD_EOM | ...
```

Cuadro 18: Segunda parte del código comentado de /usr/src/sys/scsi/scsi_base.c.

```
...
sbs = scsi_decode_sense(sense, DECODE_ASC_ASCQ);
/*if (strlen(sbs) > 0)
printf("     ASC/ASCQ: %s\n", sbs);*/
if (sense->fru != 0)
...
```

Cuadro 19: Tercera parte del código comentado de /usr/src/sys/scsi/scsi_base.c.

Script `DetectCD.sh`

La idea principal de éste *script* radica en estar pendiente cuando un disco CD es insertado en el dispositivo CD que le fue asignado. Su funcionamiento radica en la ideas que se explonen a continuación:

1. Examinar que dispositivo CD le fue asignado.

2. Verificar con ayuda del binario `cddevctl`, si existe un disco CD en el dispositivo que le fue asignado. Si no existe algún medio CD, se empieza el proceso desde el paso 2). Si existe algún medio CD, entonces el siguiente paso es determinar que tipo de CD contiene el dispositivo. Los tipos de CD, DVD pueden ser tres:

   a) *Un disco CD o DVD en blanco:* En éste caso, se registrará su entrada al sistema y se crearán los accesos al dispositivo con la llamada del *script* `CreateAccessToMedium.sh`
   Luego de ello, se estará examinando si el acceso (acceso se refiere a un icono del dispositivo) al dispositivo está aún presente en los escritorios de los usuarios. Si algún usuario ya no tiene tal acceso (se eliminó el archivo **.desktop* del dispositivo), se procederá con la eliminación de todos los accesos creados, además de su registro dentro del sistema. Hecho lo anterior, se procede a comenzar la fase de examinado desde el paso 2).

   b) *Un disco o DVD de datos:* Para éste caso, se registrará la entrada del disco CD en el sistema. Luego de ello se lanzará e ejecución al *script* `CreateAccessToMedium.sh` quien se hará cargo de crear el acceso a éste dispositivo.
   Terminado el trabajado del *script* `CreateAccessToMedium.sh`., se estará examinando si el acceso al dispositivo está aún presente en los escritorios de los usuarios. Si algún usuario ya no tiene tal acceso, se procederá con la eliminación de todos los accesos creados, su registro dentro del sistema y por último se desmontará y se expulsará el medio CD.
   Hecho lo anterior, se procede a comenzar otra vez con la fase de examinado desde el paso 2).

   c) *Un disco de audio:* Para éste caso, sólo se se registra su acceso al sistema, puesto que éste tipo de dispositivos será manejado

directamente por la aplicación más apropiada para éste tipo de medios.

En una manera resumida, la principal tarea de éste *script* es la de decidir que operaciones se deben de hacer para poder dar a los usuarios, una manera fácil de poder acceder a éste tipo de medios.

Script `CreateAccessToMedium.sh`

El objetivo de éste *script* es crear los enlaces a los dispositivos CD del sistema. Tal *script* empieza por examinar en donde va a radicar el contenido del disco CD (generalmente en un directorio dentro de `/mnt`), luego de ello determina el tipo de disco. Si es un CD o DVD de datos, se procede a su montado dentro del sistema. En caso contrario, no montará al dispositivo. Por último, se encarga de crear los accesos al dispositivo y proporcionarle a cada usuario del sistema un acceso.

En pocas palabras, nuestro *script* define la manera y proporciona los medios a los usuarios para acceder al dispositivo CD.

4.3.2. Servicio de detección de dispositivos USB

El servicio de detección de dispositivos **USB** nace desde cero. Dentro de **PC-BSD** tal servicio es dado por **HAL**. Al igual que en los medios CD, éste se centra en proporcionar un método para dar acceso y control a los dispositivos **USB** de los usuarios.

La idea básica de éste servicio se define como lo siguiente: se debe detectar el momento en el cual un dispositivo **USB** es introducido en el sistema, se debe dar acceso al dispositivo a los usuarios y se debe poder desechar al dispositivo cuando el usuario lo crea conveniente.

En nuestro sistema, éste servicio es implentado a través de 5 *scripts*, cada uno destinado a realizar una tarea muy específica en la detección de dispositivos **USB**. Los *scripts* son `SecureDetectionUSB.sh`, `DetectUSBDevices.sh`, `ScanUSB.sh`, `AgenteUSB.sh` y `CreateAccessToMedium.sh`.

Script `SecureDetectionUSB.sh`

Es el principal *script* y se encarga de inicializar todo el servicio al llamar a ejecución a `DetectUSBDevices.sh` quien empezará la detección de dispositivos **USB**.

Script `DetectUSBDevices.sh`

El trabajo de éste *script* consiste en crear un archivo de datos (al archivo `usb-encontrados.dat`) y a partir de eso, estar escaneando cada determinado tiempo al sistema con el fin de encontrar la presencia de nuevos dispositivos **USB**. Para llevar a cabo el proceso de escaneado se ejecuta al *script* **ScanUSB.sh**.

Script `ScanUSB.sh`

El *script* es el corazón del sistema de detección. Dentro de éste se centran la mayor parte de las operaciones importantes. Es por ello que su funcionamiento se define en los siguientes puntos:

1. La primera fase es examinar con que dispositivos cuenta el sistema. Ello se realiza con ayuda del comando `usbdevs`[8] mandando el resultado al archivo `usb-encontrados.dat` (Sólo se ingresan memorias **USB** a éste archivo). Los dispositivos **USB** soportados por el *script* son los pertenecientes a las familias **Data Traveler**, **Flash Disk** y **USB Drive**. Pero puede agregarse otro tipo de familia que no aparezca en la lista, siempre y cuando el *kernel* de **OpenBSD** las soporte.

2. El segundo paso es ver los resultados de `usb-encontrados.dat`. Si éste archivo tiene un tamaño igual a cero bytes, entonces no hay ningún dispositivo **USB** conectado al sistema. De no ser así, se determina el número de dispositivos **USB** en `usb-encontrados.dat` y a partir de ello, se entrará en un ciclo en donde se examinará cada registro del archivo.

3. En la fase de examinado, se determina si tal memoria **USB** no existe en los registro del sistema. Las memorias **USB** se registran en el directorio `Dispositivos-USB/` ubicado en `/tmp/X-BSD/SecureUSB/`. Dentro

[8]El comando `usbdevs` permite examinar que dispositivos **USB** estan conectados al sistema.

de dicho directorio se crean archivos de texto con el nombre del dispositivo asignado por `usbdevs`). En caso de existir un registro de la memoria en el sistema, se pasa a examinar la siguiente memoria **USB** encontrada. En caso contrario se crea el registro de la memoria **USB** en el directorio `Dispositivos-USB`. Por último, se manda a ejecutar el *script* `AgenteUSB.sh`, quien se encargará del estado de la memoria hasta que sea retirada del sistema.

En cierto modo éste *script* trabaja como un servidor de memorias **USB**, y en el momento en que alguna es detectada se le asigna un subproceso que atienda todas las peticiones solicitadas a la memoria **USB**.

Script `AgenteUSB.sh`

Tal *script* simboliza un agente encargado del estado de una memoria **USB**. Sus principales operaciones son determinar todos los datos necesarios de la memoria **USB**. Los datos más comunes son el tipo, la marca, su tamaño, su principal partición, el nombre del nodo asignado y muchos más.

Cuando se tienen todos los datos necesarios, se lanza a ejecución al *script* `CreateAccessToMedium.sh` quien se encargará de proporcionar los accesos a éste dispositivo a todos los usuarios del sistema. Terminada la ejecución del *script* anterior, se creará el registro del dispositivo en el directorio `Dispositivos-USB` pasando así a una fase de examinado, en donde determinado tiempo se examinará si alguno de todos los accesos creados a la memoria **USB** ya no esta presente.

En el caso de no estar presente, se termina la fase de examinado, se desmonta la memoria **USB** y para cada usuario en el sistema se elimina tal acceso comenzando así una fase de espera en donde el proceso terminará en el momento que se retire tal dispositivo.

Script `CreateAccessToMedium.sh`

Por último, éste *script* se encarga de crear un directorio dentro de `/mnt` en donde radicará en contenido de la memoria **USB** con ciertos permisos de escritura para los usuarios de sistema.

Luego de ello, será montada la memoria **USB** y se empezará a configurar un enlace al dispositivo con ayuda de un archivo **.desktop* en base a la especificación de accesos directos de **KDE**. Tal enlace se copiará a cada uno de los usuarios para que éstos puedan utilizar a la memoria **USB**. En el Cuadro 20, se puede ver un ejemplo sobre el contenido de un acceso a una memoria **USB** la cual radica en `/mnt/Usb-sdx`.

```
[Desktop Entry]
Encoding=UTF-8
Type=Link
URL=/mnt/Usb-sdx
```

Cuadro 20: Archivo de configuración **.desktop* para memorias **USB**.

En cortas palabras, así se estructuran las operaciones sobre las memorias **USB** en el sistema **X-BSD**.

Capítulo 5

Conclusiones y perspectivas a futuro

5.1. Conclusiones

El sistema **X-BSD** en su primera versión es configurado con una presentación similar a **PC-BSD**. Nuestro sistema da acceso a un estorno de escritorio **KDE** por ser la interfaz más amigable que se puede usar en éste tipo de sistemas.

Otro punto importante es el hecho de las comparaciones con otros trabajos realizados con **OpenBSD** con enfoque a usuarios finales. Hasta el día de hoy, no es posible hacer una comparación entre los diferentes proyectos como **Anonym.OS**[1], **Gentoo/OpenBSD**[2], **Quetzal**[3] y **OliveBSD**[4], ya que cada uno de ellos está destinado a cubrir objetivos diferentes a los desarrollados sobre **X-BSD** en su primera versión.

[1]El **Live CD** de **Anonym.OS** provee un acceso a la **Web** de manera anónima a todo tipo de usuarios, haciendo que una máquina corriendo éste sistema sea visto como un sistema **Windows XP**.

[2]El **Live CD Gentoo/OpenBSD** tiene la meta de crear un completo sistema **Gentoo** basado en **OpenBSD**, se pretende compartir las facilidades de administración de **Gentoo** con la confiabilidad del kernel de **OpenBSD**.

[3]El **Live DVD** de **Quelzal** cuenta con una instalación completa de **OpenBSD** y está enfocado a contar con algunas aplicaciones selectas para cálculos matemáticos.

[4]El **Live CD** de **OliveBSD** está basado en OpenBSD 3.8, cuenta con un ambiente gráfico con software ya instalado.

5.2. Perspectivas a futuro

Durante las últimas semanas de dicembre y parte de enero, se han revelado nuevas características y funcionalidades que harían al sistema más atractivo. Pero como es de esperarse, tales ideas quedan fuera del marco de trabajo expuesto en la tesis. Algunas de ellas, se explican a continuación:

- Se podría pensar en la posibilidad de contar con un instalador gráfico, puesto que la forma de instalación actual requiere que el usuario sepa instalar **OpenBSD**. Ello debido a que la fase de instalación se **X-BSD** durante el primer inicio del sistema.

- Otra característica sería el tener una mayor cantidad de *directorios-app* del *sistema de puertos*. En la actualidad sólo se cuentan con dos aplicaciones migradas las cuales son: **mozilla-firefox** y **mozilla-thunderbird**. Con respecto a los *directorios-app*, se podría desarrollar una mejor integración de ellos con el sistema. Es decir, el sistema debe poder definir qué *directorio-app* está destinado a manejar un tipo de archivo específico de manera automática.

- Con respecto a los paquetes de **OpenBSD**, se podría también tratar portar **HAL** al *sistema de puertos* como lo tiene **FreeBSD**.

- En cuanto a la internacionalización del sistema, se podría desarrollar con soporte para más idiomas tanto para los casos del instalador, el entorno gráfico **KDE** y los *directorios-app*. Todo ello, puesto que en éste momento sólo se soporta el idioma Ingles dentro de **KDE** y en los *directorios-app*.

- Por último, pensando en características más ambiciosas, se podría ver la posibilidad de llevar a **KDE** a ser simplemente un *directorio-app*. Otra posibilidad sería el contar con un núcleo "invariante", es decir, que no dependa del entorno gráfico eligido (en particular poder incluir **GNOME**, **GNUStep**, entre otros).

Bibliografía

[1] Paco Hope, Yanek Korff, Bruce Potter, *Mastering FreeBSD and OpenBSD Secury*, O'Reilly, March 2005

[2] Michael W. Lucas, *Absolute OpenBSD: UNIX for the Practical Paranoid*, No Starch Press, San Francisco California, EU, 2nd Edition, 2003.

[3] Michael W. Lucas, *Absolute FreeBSD: The complete guide to FreeBSD, 2nd Edition*, No Starch Press, San Francis7co California, EU, 2nd Edition, Nov 2007.

[4] Michael W. Lucas and Jordan Hubbard, *Absolute BSD: The complete guide to FreeBSD, 2nd Edition*, No Starch Press, San Francisco California, EU, 2nd Edition, August 2002.

[5] Marshall Kirk McKusick and George V. Neville, *The Design and Implementation of the FreeBSD Operating System*, Addison-Wesley Professional 2nd Edition, Copyrigth 2005.

[6] Dru Lavigne, *BSD Hacks*, O'Reilly, 2004.

[7] Micho Durdevich, *Installing BSD Operating Systems On IBM Netvista S40*, `http://www.daemonnews.org/`, 2005.

[8] Jopeph Kong, *Designing BSD Rootkits*, O'Reilly, First Edition April 2007.

[9] Brandon Palmer, Jose Nazario, *Secure Architectures with OpenBSD*, O'Reilly,April 2004

[10] The OpenBSD project, `http://www.openbsd.org/`

[11] The FreeBSD project, `http://www.freebsd.org/`

[12] The NetBSD project, http://www.netbsd.org/

[13] Licencia BSD,
http://www.freebsd.org/copyringth/freebsd-licence.html

[14] Licencia GSD, http://es.wikipedia.org/wiki/GNU_GPL

[15] Quetzal: A Live OpenBSD System, http://quetzal.matem.unam.mx/

[16] OliveBSD-OpenBSD Live CD,
http://g.paderni.free.fr/olivebsd/

[17] Anonym.OS LiveCD,
http://sourceforge.net/projects/anonym-os/