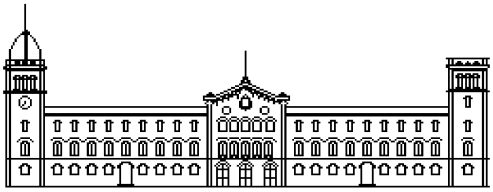
UNIVERSITAT DE BARCELONA



Treball fi de carrera

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques

Universitat de Barcelona

Análisis, configuración y administración de sistemas operativos basados en Unix

---

Aníbal Moreno Gil

Director: Sergio Escalera
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 20 de juliol de 2004

Resumen

En el mundo de la informática una pieza clave para su buen funcionamiento y desarrollo son los sistemas operativos donde se ejecutan todos nuestros programas. Estos sistemas operativos facilitan una plataforma de ejecución organizada y estable. De este modo, los programadores no tienen que preocuparse del hardware donde se ejecutará su software. Por otro lado, la gran demanda de sistemas operativos y la evolución de la informática ha llevado a los desarrolladores a diversificar las ramas para adaptarlos a los nuevos tiempos y, aunque el mundo de la informática no tenga una de las historias más extensas, el número de sistemas operativos diferentes es ya considerable.

En el proyecto, nos centraremos en dos de estos sistemas y realizaremos un estudio comparativo. Descubriremos la historia de los sistemas basados en Unix y, concretamente, la de los dos sistemas seleccionados para el proyecto: Debian y OpenSolaris. Además, veremos las características más destacables de cada uno, así como una comparativa entre los rendimientos de cada uno de los sistemas mediante unos test manufacturados. Por último, investigaremos una parte imprescindible en todo sistema operativo, los sistemas de almacenamiento.

Resum

En el món de la informàtica una peça clau pel seu bon funcionament i desenvolupament són els sistemes operatius on s'executen tots els nostres programes. Aquests sistemes operatius faciliten una plataforma d'execució organitzada i estable pels nostres programes. Per una altra banda, la gran demanda de sistemes operatius i la evolució de la informàtica han portat als desenvolupadors a diversificar les branques per adaptar-los als nous temps i, encara que el món de la informàtica no tingui una de les històries més extenses, el número de sistemes operatius diferents, ara per ara, és considerable.

En el projecte, ens centrarem en dos d'aquests sistemes i realitzarem un estudi comparatiu. Descubrirem la història dels sistemes basats en Unix i, en concret, la dels dos sistemes selecionats pel projecte: Debian i OpenSolaris A més a més, veurem les característiques més destacables de casascun, així com una comparatica entre els rendiments de cada un dels sistemes per mitjà d'uns test manufaturats. Per últim, investigarem una part imprescindible en tot sistema operatiu, els sistemes d'emmagatzematge.

Abstract

A key in the world of computing for its proper functioning and development are operating systems. These operating systems provide a platform for an organized and stable performance of our programs. In this way, programmers do not have to worry about the hardware where they run their software. On the other hand, the high demand for operating systems and the evolution of computers has led the developers to diversify its branches to adapt to new times. Although the computer world does not have one of the longest histories, the number different operating systems is already considerable.

In this project, we focus on two of these systems and perform a comparative study. We describe the history of Unix-based systems and, specifically, of two systems selected for the project. In addition, we will analyze the highlights of each, and a comparison between the yields of each of the systems manufactured by a test. Finally, we will investigate an indispensable part in any operating system, the storage systems.

Índice

1. Introducción

Desde que descubrí el mundo de la informática he experimentado una gran curiosidad por todas las partes que lo componen: desde el hardware hasta los programas más complejos, pasando, como no, por Internet. Desde mis inicios he profundizado mucho más en el diseño e implementación de programas en diversos lenguajes de programación que en ningún otro campo, ya que el mundo de la informática está muy enfocado a este objetivo. Hasta que un día me pregunté: ¿qué sé del entorno donde se ejecutan mis programas? Entonces empecé investigar sobre el hardware, pero descubrí que existía un gran salto entre lo que yo hacía con mis programas y lo que realmente realizaba el hardware de mi PC.

La siguiente pregunta que me formulé fue: ¿qué hace que mis programas pueden ejecutarse en el hardware de mi PC? La clave que lo unía todo era el sistema operativo. Con él se logra establecer un flujo de comunicación constante y fiable entre el hardware más básico y nuestros programas, por complejos que éstos sean.

Desde la creación de los primeros sistemas operativos, la oferta de los mismos se ha visto incrementada exponencialmente, se han diversificados sus ramas y se han creado nuevos. Algunos se han hecho muy comunes entre los entornos de escritorio y otros se han especializado más en entornos empresariales, pero todos siguen teniendo un denominador común, facilitar a nuestros programas un entorno de ejecución completo sobre el hardware existente.

2. Objetivos

En este proyecto me gustaría investigar en profundidad los sistemas operativos más utilizados en la actualidad, además de vislumbrar las diferencias entre sus diversos diseños. Para ello, analizaré su comportamiento en diferentes entornos y situaciones, así como observar sus diferentes configuraciones e instalaciones.

Realizaré un estudio comparativo entre dos de las grandes familias más diferenciadas basadas en UNIX: Linux y Solaris. Para dicho estudio utilizaré sus versiones OpenSource: Debian, y OpenSolaris.

Además, me gustaría investigar la historia y evolución de cada uno de ellos para poder tener una visión con la suficiente perspectiva del entorno en el que han germinado cada uno de ellos. Por otro lado, realizaremos un estudio de las características de cada sistema y realizaremos unos test de rendimiento para vislumbrar la capacidad de cada uno de resolver los problemas que aparecen durante la ejecución de nuestros programas.

También he añadido otro aparatado importante en nuestros sistemas operativos y ligados directamente con ellos, los sistemas de almacenamiento. Actualmente los sistema operativos no gestionan directamente los dispositivos físicos, sino que delegan esta función a sistema de almacenamiento que permiten administrar de forma eficaz nuestros dispositivos hardware.

Por último, indicar que durante la evolución del proyecto han surgido numerosas inquietudes tale como los Runlevels, las zonas de OpenSolaris, el DTrace y, dentro de los posible, he intentado satisface mi curiosidad.

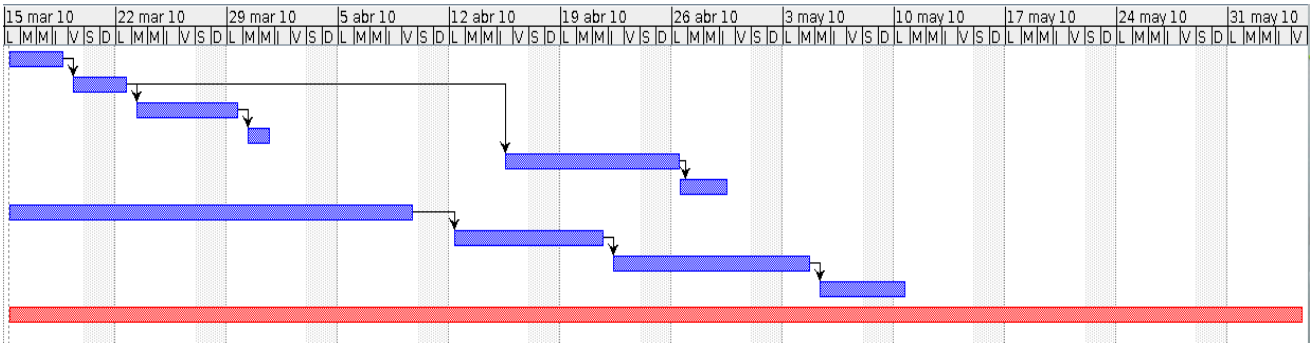
3. Planificación y costes

Los costes para realizar el proyecto, en cuanto a hardware, han sido nulos ya que para las pruebas he utilizado tanto mi portátil personal como un PC de sobremesa, aunque su costes son 900€ de portátil y 500€ del PC.

Las tareas siguientes son las que he realizado para la realización de este proyecto. La parte de sistemas de almacenamiento la he realizado entre una máquina virtual dentro de portátil para disponer de numerosos disco virtuales y en PC. En cuanto a las pruebas de rendimiento, las he realizado completamente sobre el hardware del PC para eliminar capas lógicas y obtener una pruebas más realistas. En la siguiente imagen indicamos una fechas aproximadas de cada una de las tareas realizadas:

	Nombre	Descripción	Duración	Inicio	Terminado
1	Instalación SO base	Crear el SO base	4 days	15/03/10 8:00	18/03/10 17:00
2	Instalación y configuración Virtualbox	Instalar VirtualBox, crear discos virtuales y configurar los dispositivos a compartir	2 days	19/03/10 8:00	22/03/10 17:00
3	Instalalación Debian	Instalar la útlima version estable de Debian y actulizar los últimos parches	5 days	23/03/10 8:00	29/03/10 17:00
4	LVM	Instalar y configurar el LVM en Debian	2 days	30/03/10 8:00	31/03/10 17:00
5	Instalación OpenSolaris	Instalar la útlima versión de OpenSolaris y actulizar los últimos parches	7 days	15/04/10 13:00	26/04/10 13:00
6	ZFS	Instalar y configurar el ZFS	3 days	26/04/10 13:00	29/04/10 13:00
7	Diseño de brenchmark	Diseñar e implementar los Brenchmark, además de realizar pruebas de ejecución	20 days	15/03/10 8:00	9/04/10 17:00
8	Configuración pre-requisitos brenchmark	Instalar todo lo requisitos para la ejecución de las pruebas	8 days	12/04/10 8:00	21/04/10 17:00
9	Brenchmark	Ejecución de las pruebas	9 days	22/04/10 8:00	4/05/10 17:00
10	Análisis de los datos obtenidos	Analizamos y extraemos las conclusiones de las pruebas	4 days	5/05/10 8:00	10/05/10 17:00
11	Documentación	Elavoración de la documentación	60 days	15/03/10 8:00	4/06/10 17:00

Además, este es el diagrama de Grantt que hemos utilizado para la realización del proyecto:



El hardware del portátil utilizado es el siguiente:

- ✔ Procesador: Intel © Core™2 Duo CPU T9300 @ 2.50GHz
- ✔ Placa base: Chipset Intel® 965PM Express
- ✔ Memoria: 4 GB DDR2
- ✔ Tarjeta gráfica: NVIDIA® GeForce® Go 8600M GT 512MB
- ✔ Disco duro: SATA Seagate de 320 Gigabytes

En cuanto al PC de sobremesa, el hardware es el siguiente:

- ✔ Procesador: Intel © Core™2 Duo CPU 9300 @ 1.86GHz
- ✔ Placa base: Asus P5B con chipset Intel P965 Express
- ✔ Tarjeta gráfica: Asus Radeon HD 5850
- ✔ Memoria: 2 GB DDR2
- ✔ Discos duros: dos discos SATA Seagate Barracuda 7200.10 de 80 Gigabytes

4. Sistemas operativos

En este proyecto nos hemos centrado en una de las grandes familias de sistemas operativos: Unix. Es un sistema operativo desarrollado, en principio, en 1969 por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy y Joe Ossanna [1].

Las principales características comunes de la familia Unix son:

i. Portabilidad: Propiedad de los sistemas operativos que permite la ejecución de los mismos en diferentes plataformas. A mayor portabilidad menor es la dependencia del sistema con respecto a la plataforma [2].
ii. Multitarea: Capacidad que permite que varios procesos sean ejecutados al mismo tiempo compartiendo uno o más procesadores [3].
iii. Multiusuario: Característica de un sistema operativo que permite la concurrencia de más de un usuarios compartiendo todos los recursos del sistema[4].

Existen varias familias del sistema operativo UNIX, que han evolucionado de manera independiente a lo largo de los años. Las más significativas son:

- BSD: familia originada por el licenciamiento de UNIX a Berkeely. Se reescribió para no incorporar propiedad intelectual originaria de AT&T en la versión 4. La primera implementación de los protocolos TPC/IP que dieron origen a Internet son la pila (stack) TCP/IP BSD.

- AIX: Esta familia surge por el licenciamiento de UNIX System III a IBM.

- GNU: En 1983, Richard Stallman anunció el Proyecto GNU, un proyecto para crear un sistema similar a Unix, que pudiese ser distribuido libremente. El software desarrollado por el proyecto (por ejemplo, GNU Emacs y CGG) también han sido parte fundamental de otros sistemas UNIX.

- Linux: En 1991, cuando Linus Tolvards empezó a proponer un núcleo Linux y a reunir colaboradores, las herramientas GNU eran la elección perfecta. Al combinarse ambos elementos, conformaron la base del sistema operativo (basado en POSIX) que hoy se conoce como GNU/Linux. Las distribuciones basadas en el núcleo, el software GNU y otros agregados entre las que se pueden mencionar a Red Hat Linux y Debian GNU/Linux se han hecho populares en los entornos empresariales.

Por último, me gustaría hablar de las implementaciones comerciales de Unix más importantes hasta ahora. La gran mayoría de entornos empresariales utilizan en sus servidores alguna de las siguientes implementaciones:

- Solaris: Desarrollado desde 1992 por Sun Microsystems y, actualmente, por Oracle Corporation. Es una de las distribuciones más estables y con mejor rendimiento. Utilizadas sobre todo para servidores de bases de datos.
- De entre la numerosas distribuciones de Linux:
  - Red Hat Enterprise Linux: Distribución comercial desarrollada por Red Hat. Una de sus virtudes es que tiene un soporte oficial de Red Hat y programas de certificación.
  - Debian: Una de las distribuciones con más apoyo entre la comunidad de desarrolladores y usuarios y basado completamente en software libre.
  - Suse Linux: Una de las versiones más sencillas de instalar y administrar, ya que cuenta con varios asistentes gráficos para completar diversas tareas en especial por su gran herramienta de instalación y configuración YasT.
- HP-UX: es la versión de Unix desarrollada y mantenido por Hewlett-Packard desde 1983. Sus mejores características son su gran estabilidad y escalabilidad, en parte gracias a sus sistemas de virtualización propios de HP (HP Serviceguard).

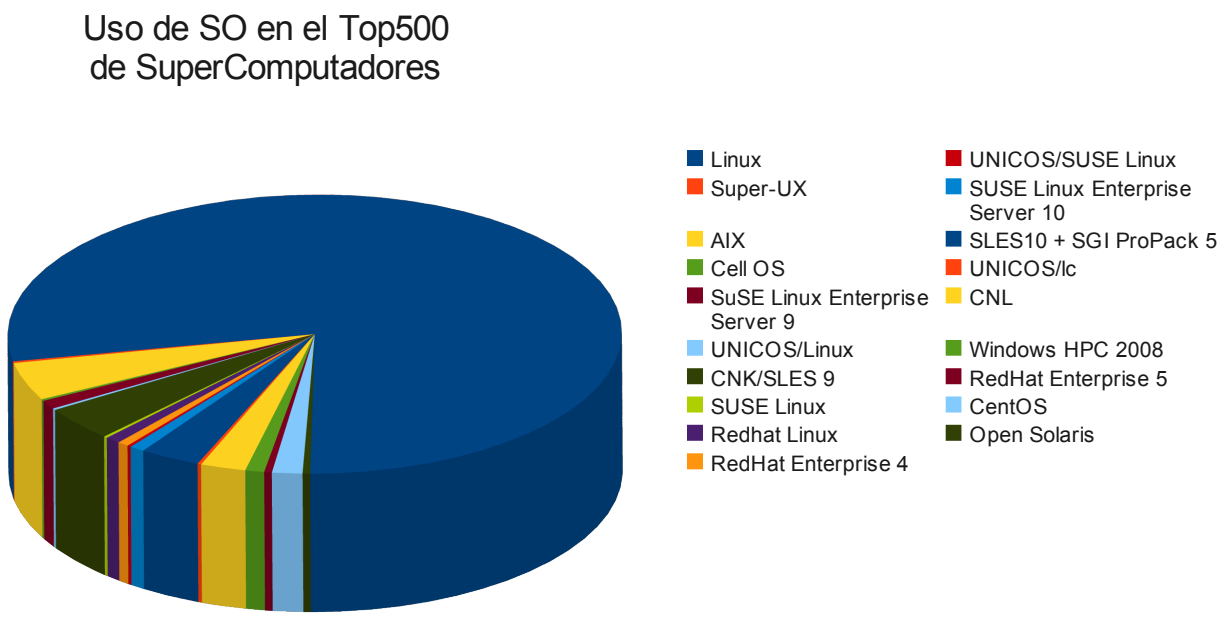
Llegados a este punto, pasaremos a hablar de estadísticas de uso en el mundo real de los sistemas que hemos nombrado. Como podremos ver en la estadística siguiente existen 19 sistemas operativos diferentes en los 500 supercomputadores más potentes de la actualidad. Pueden parecer muchos pero si observamos con atención podemos apreciar que el 96% utilizan tan sólo 5 sistemas operativos diferentes.

El más utilizado, y con gran diferencia, es Linux. Seguido a gran distancia por AIX, CNK/SLES 9, SLES10 + SGI ProPack5 y CNL [5].

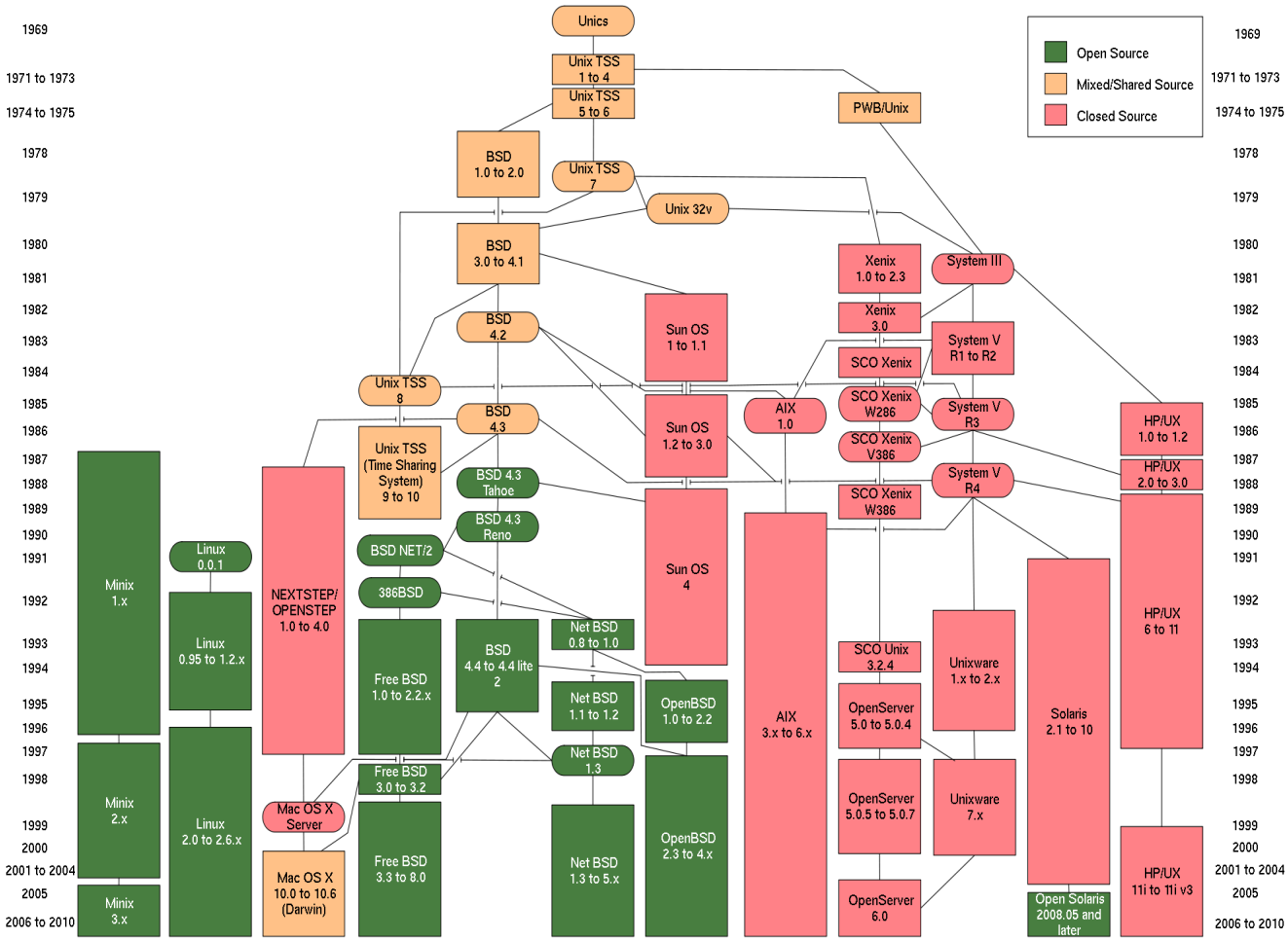
Operating System	Count	Share %
Linux	391	78.20 %
Super-UX	1	0.20 %
AIX	22	4.40 %
Cell OS	1	0.20 %
SuSE Linux Enterprise Server 9	5	1.00 %
UNICOS/Linux	1	0.20 %
CNK/SLES 9	20	4.00 %
SUSE Linux	1	0.20 %
Redhat Linux	4	0.80 %
RedHat Enterprise 4	3	0.60 %
UNICOS/SUSE Linux	1	0.20 %
SUSE Linux Enterprise Server 10	4	0.80 %
SLES10 + SGI ProPack 5	16	3.20 %
UNICOS/lc	1	0.20 %
CNL	12	2.40 %
Windows HPC 2008	5	1.00 %
RedHat Enterprise 5	2	0.40 %
CentOS	8	1.60 %
Open Solaris	2	0.40 %

Cabe destacar que de los 5 primeros supercomputadores, 4 son versiones de Linux. Esto no significa que sean los que mejor rendimiento obtenga o que más estables sean, sino que al ser de código abierto los ingenieros de sistemas tiene la posibilidad de modificar a placer su sistema operativo hasta adaptarlo al máximo a sus objetivos. Otra de las posibles causas suele ser la reducción del gasto en licencias de software, ya que abarata los gastos del proyecto en gran medida.

A continuación, se muestran las estadísticas anteriores en un gráfico :



Por último, me gustaría incluir un diagrama con todas la familias descendientes de Unix. Como podemos observar las ramas cada vez se diversifican más, sobre todo gracias a la comunidad de desarrolladores de código libre. Por un lado, tenemos los sistemas operativos impulsados por grandes corporaciones como HP-UX (de Hewlett-Packard), AIX de IBM, Solaris y Sun OS actualmente propiedad de Oracle Corp. (antes Sun Microsystems), Mac OS X de Apple. Y, por otro, los sistemas de código libre, como la familia BSD (FreeBSD, NetBSD y OpenBSD), de reciente creación como OpenSolaris y las más antiguas y maduras como Linux y Minix [6].


1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2010
Open Source
Mixed/Shared Source
Closed Source
Unics
Unix TSS 1 to 4
Unix TSS 5 to 6
Unix TSS 7
Unix 32v
PWB/Unix
BSD 1.0 to 2.0
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
BSD 4.3 Tahoe
BSD 4.3 Reno
Unix TSS 8
Unix TSS (Time Sharing System) 9 to 10
BSD NET/2
386BSD
Free BSD 1.0 to 2.2.x
Free BSD 3.0 to 3.2
Free BSD 3.3 to 8.0
BSD 4.4 to 4.4 lite 2
Net BSD 0.8 to 1.0
Net BSD 1.1 to 1.2
Net BSD 1.3
Net BSD 1.3 to 5.x
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 4.x
AIX 1.0
AIX 3.x to 6.x
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix W286
SCO Xenix V386
SCO Xenix W386
SCO Unix 3.2.4
OpenServer 5.0 to 5.0.4
OpenServer 5.0.5 to 5.0.7
OpenServer 6.0
System III
System V R1 to R2
System V R3
System V R4
Unixware 1.x to 2.x
Unixware 7.x
Solaris 2.1 to 10
Open Solaris 2008.05 and later
HP/UX 1.0 to 1.2
HP/UX 2.0 to 3.0
HP/UX 6 to 11
HP/UX 11i to 11i v3
Minix 1.x
Minix 2.x
Minix 3.x
Linux 0.0.1
Linux 0.95 to 1.2.x
Linux 2.0 to 2.6.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.6 (Darwin)

4.1 Casos de estudio

Como hemos podido observar con lo visto hasta ahora la familia de sistemas Unix es muy extensa y diversa. Por este motivo no podemos hacer un estudio de cada una de ellas. Nos centraremos en dos de las ramificaciones más estables y utilizadas, Linux y Solaris. Concretamente, en la versión bajo licencia CDDL (código abierto y libre) OpenSolaris y en la distribución Debian de Linux. Es importante asumir que existen muchos elementos comunes entre ambos sistemas puesto que tienen la misma raíz, por lejana que ésta sea.

Sin más preámbulos, pasamos a realizar un estudio de estos dos sistemas, daremos unas breves pinceladas a la historia de cada uno de los sistemas operativos que utilizaremos así como sus versiones y características especiales.

4.2 Linux

Linux es el núcleo del sistema operativo libre, llamado con el mismo nombre, descendiente de Unix. Es usualmente utilizado junto a las herramientas GNU como interfaz entre los dispositivos hardware y los programas usados. A la unión de ambas tecnologías, incluyendo entornos de escritorio e interfaces gráficas, se las conocen como distribución GNU/Linux. Fue lanzado bajo la licencia pública general de GNU y es desarrollado gracias a contribuciones provenientes de colaboradores de todo el mundo, por lo que es uno de los ejemplos más notables de software libre. Debido a su naturaleza de contenido libre, ambos proyectos invitan a colaborar en ellos de forma altruista.

4.2.1 Historia

Linux fue creado por Linus Torvalds en 1991. Tras ser publicado, la comunidad de Minix (muy fuerte durante aquella época) contribuyo en el código y en ideas para el núcleo. A pesar de las funcionalidades limitadas de la primer versión, a pesar de la unión con el proyecto GNU, rápidamente fue acumulando desarrolladores y usuarios que adoptaron el código para crear nuevas distribuciones [7].

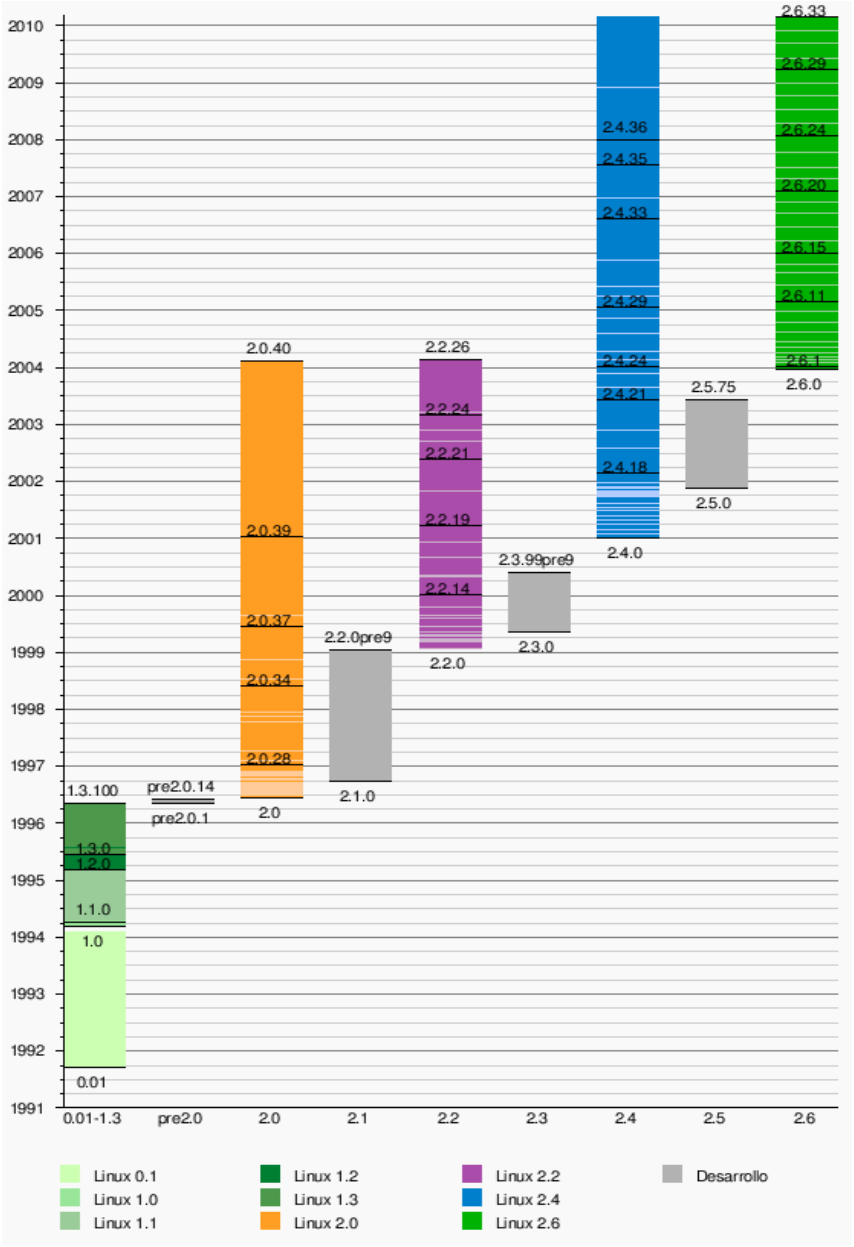
Hoy en día, el núcleo Linux ha recibido contribuciones de miles de programadores. Es el tercer sistema operativo más utilizado en entornos de escritorio y el más utilizado en servidores.

Cronología:

- ✔ En septiembre de 1991 se lanzó la versión 0.01 de Linux. Tenía 10.239 líneas de código.
- ✔ En diciembre se lanzó la versión 0.11 que ya era auto albergada.
- ✔ 0.95 fue la primera versión capaz de ejecutar la implementación de X Windows System (Xfree86).

- ✔ En marzo de 1994, se lanzó la versión 1.0.0, que constaba de 176.250 líneas de código.
- ✔ En marzo de 1995, se lanzó la versión 1.2.0, que constaba de 310.850 líneas de código.
- ✔ La versión 2 fue lanzada en junio de 1996 con gran éxito.
- ✔ En enero de 1999 se lanzó la versión 2.2.0 con 1.800.847 líneas de código.
- ✔ En diciembre del mismo año, se publicaron parches de IBM Mainframe para 2.2.13, permitiendo así que Linux fuera usado en ordenadores corporativos.
- ✔ En enero de 2001 se lanzó la versión 2.4.0 con 3.377.902 líneas de código.
- ✔ En diciembre de 2003 se lanzó la versión 2.6.0 con 5.929.913 líneas de código.
- ✔ En el mismo mes de 2008 se consiguió alcanzar las 10.195.402 líneas de código con la versión 2.2.28.

A continuación se muestra una imagen con la cronología anterior [8]:



En cuanto a la distribución que vamos a utilizar, Debian, pertenece al Proyecto Debian, que es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre. El modelo de desarrollo del proyecto es ajeno a motivos empresariales o comerciales, siendo llevado adelante por los propios usuarios, aunque cuenta con el apoyo de varias empresas de forma de infraestructuras [9].

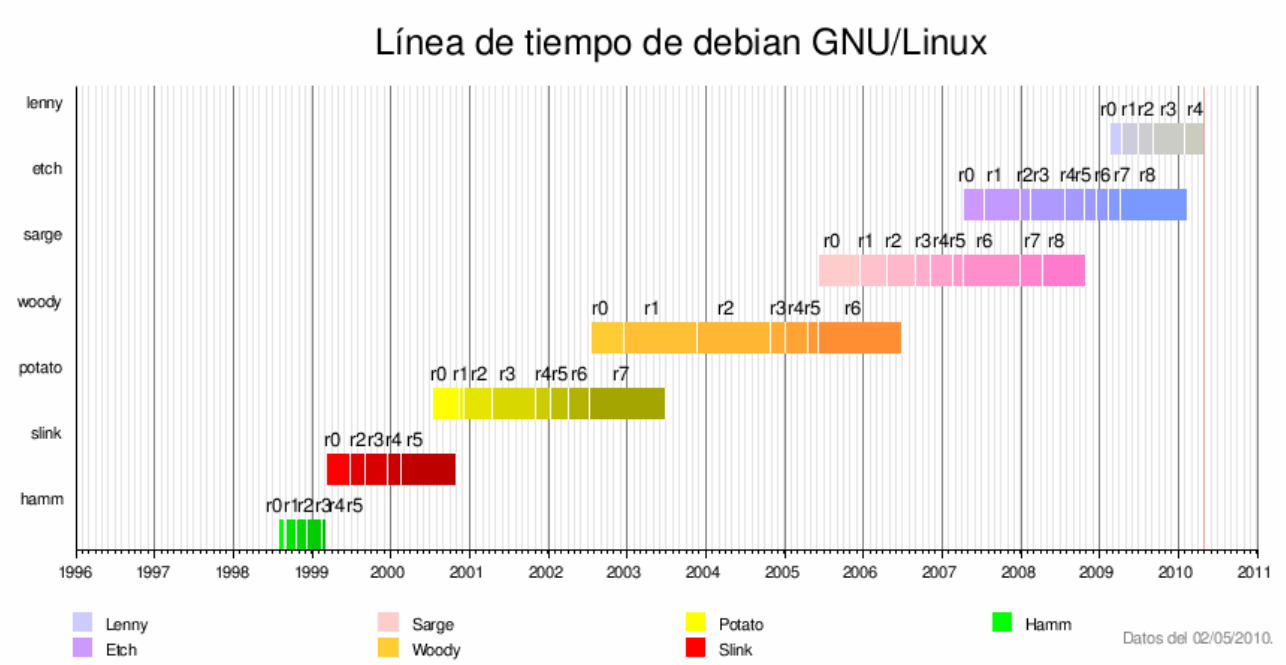
Fue fundado en el año 1993 por Ian Murdock, después de haber estudiado en la Universidad de Purdue. Él escribió el manifiesto de Debian que utilizó como base para la creación de la distribución Linux Debian. Dentro de este texto los puntos destacables son: mantener la distribución de manera abierta, coherente al espíritu de Linux y de GNU.

El sistema se encuentra precompilado, empaquetado y en un formato sencillo para múltiples arquitecturas de computador y para varios núcleos.

Cada una de las versiones que se muestran en el cuadro siguiente pasa por los estados siguientes:

- Estable: Cuenta con el apoyo del equipo de seguridad de Debian y e la recomendada para uso en producción.
- Testing: En esta versión se encuentran paquetes que han estado previamente en ella versión inestable, pero que contienen mucho menos fallos.
- Inestable: Esta versión es donde tiene lugar le desarrollo activo de Debian. Es la rama que usan los desarrolladores del proyecto.
- Congelada: Cuando la versión de pruebas llega a un nivel aceptable de fallos, entonces se "congela", lo que significa que ya no se aceptan nuevos paquetes desde la versión inestable. A continuación se trabaja para pulir el mayor número de bugs posibles, para así liberar la versión estable.

Por último, el gráfico siguiente muestra una línea temporal de las versión de Debian:



4.2.2 Arquitectura

El kernel de Linux, y por extensión el de Debian, es del tipo monolítico. Esto quiere decir que el núcleo es complejo y de gran tamaño, englobando todos los servicios del sistema. Tiene un mayor rendimiento respecto a los del tipo micronúcleo. Con la incursión del sistema de módulos ejecutables en tiempo de ejecución se ha eliminado una de las grandes desventajas de los núcleos monolíticos, la necesidad de recompilación del núcleo y reinicio del sistema para aplicar los cambios realizados sobre cualquiera de los servicios del sistema [10].

Otra característica a destacar es que el núcleo del sistema concentra todas las funcionalidades posibles (planificación, sistema de archivos, redes, controladores de dispositivos, gestión de memoria, etc...) dentro de un gran programa. Todos los componentes funcionales del núcleo tienen acceso a todas las estructuras de datos internas y a sus rutinas. Un error en una rutina puede propagarse a todo el núcleo.

4.2.3 Jerarquía de directorios

La jerarquía de directorios es una norma que define los directorios principales y sus contenidos en sistemas de la familia Unix. Se diseñó originalmente en 1994 para estandarizar el sistema de archivos de las distribuciones Linux, basándose en la tradicional organización de directorios de los sistemas Unix [11].

Concretamente el sistemas Linux, se sigue la siguiente estructura:

- Estáticos: Contienen archivos que no cambian sin la intervención del administrador, sin embargo, pueden ser leídos por cualquier usuarios del sistema. Ejemplos: /bin, /sbin, /usr/bin, etc.
- Dinámicos: Contienen archivos que pueden crecer. Por esta razón, es recomendable que estos directorios estén montados en una partición diferente, evitando así que llenen el filesystem y causen posibles incidencias. Ejemplos: /var/tmp, /var/mail, /var/spool, etc.
- Restringidos: Normalmente, contienen directorios/ficheros de configuración del sistema y sólo son accesibles/modificables por el administrador del sistema.

A continuación nombraremos algunos de los directorios, además de su función dentro del sistema de directorios:

- / → Directorio raíz, contenedor de toda la jerarquía de ficheros.
- /bin → Directorio donde se almacenan las aplicaciones binarios, es decir, los comandos esenciales par que estén disponibles para todos los usuarios del sistema.
- /boot → Directorio donde se conservan los ficheros utilizados en el arranque del sistema.
- /dev → Este directorio contiene los portales a dispositivos del sistema, incluso a los que no se les ha asignado un directorio.
- /etc → Ficheros de configuración de todo el sistema se almacenan en este directorio.
- /home → Como su nombre indica, cada usuario tiene bajo este directorio sus propios directorios de trabajo.
- /lib → Contiene todas la librerías compartidas para todos los binarios del /bin.
- /opt → Ubicación donde se instalan aplicaciones estáticas,
- /proc → En este directorio se almacena un sistema de ficheros de texto virtuales que documentan al núcleo y el estado de los procesos que se están ejecutando en el sistema.
- /root → Directorio home del usuario root (administrador del sistema).
- /sbin → Equivalente al /bin, pero en este caso son comandos y programas que puede ejecutar, exclusivamente, el usuario root.
- /tmp → Ubicación de los ficheros temporales. Puede ser utilizado por cualquier usuario del sistema.
- /usr → Jerarquía secundaria para datos compartidos de sólo lectura. Puede ser compartido por múltiples sistemas y no debe contener datos específicos del sistema que los comparte.
- /var → Directorio donde se almacenan ficheros variables, tales como logs, spool's, temporal para el correo, etc.

4.2.4 RunLevels

Cada vez que arrancamos un sistema operativo nuestro sistema va superando diferentes etapas hasta llegar a estar totalmente disponible para su utilización. En la primera de ellas, se inicia la BIOS donde se chequea todo el hardware. La siguiente para por el gestor de arranque y, posteriormente, el kernel de Linux y, por último, el proceso que arranca todos lo servicios [12].

En esta última etapa entra en juego el proceso init. Este proceso establece el entorno de usuario, verifica y monta los sistemas de archivos, inicia los procesos denominados “demonios” y se ejecutan unos scripts organizados por un sistema denominados runlevels.

Los runlevels son una característica común entre los sistemas Unix System V para diferenciar los diferente modos de operación de nuestro sistema. Existen 7 niveles y cada uno tiene sus propios scripts los cuales involucran un conjunto de programas. Estos scripts se guardan en directorios con nombres como "/etc/rc...". El archivo de configuración de init es /etc/inittab.

Cuando el sistema se arranca, se verifica si existe un runlevel predeterminado en el archivo /etc/inittab, si no, se debe introducir por medio de la consola del sistema. Después se procede a ejecutar todos los scripts relativos al runlevel especificado.

Los niveles de ejecución estándar son lo siguientes:

(0) Reservado para el apagado del sistema.
(1) Modo monousuario, normalmente utilizado para llevar a cabo tareas de mantenimiento y reparación.
(2) Modo multiusuario, arranca para múltiples usuarios pero sin servicios de red (NFS, Samba, etc)
(3) Modo multiusuario con soporte de red, es el nivel más utilizado. Arranca todos los servicios excepto el gráfico (X11).
(4) No usado, normalmente se utiliza como runlevel personalizable, para uso del administrador.
(5) Multiusuario con entorno gráfico (X11), este runlevel arranca todos los servicios existentes en el sistema.
(6) Reservado para el reinicio del sistema, comúnmente en este runlevel conviven todos los scripts de parada de los servicios del sistema. Usualmente, también se añaden los scripts de parada de las base de datos y los aplicativos.

4.3 Solaris

4.3.1 Historia

En este apartado, empezaremos hablando del sistema operativo Solaris y para terminar de OpenSolaris, puesto que existe una estrechar relación entre ellos y la historia de nuestro caso de estudio no se entendería adecuadamente sin explicar previamente la de Solaris [13].

Solaris es un sistema operativo de la familia Unix desarrollado por Sun Microsystemes desde 1992 y actualmente por Oracle Corporation (tras la compra de Sun Microsystemes). El primer sistema operativo de Sun, SunOs, nació en 1983. Estaba basado en el sistema BSD, de la Universidad de Berkeley aunque más adelante incorporó numerosas funcionalidades de System V, convirtiéndose prácticamente en un sistema operativo totalmente basado en System V y con un kernel monolítico. Con posteridad se distingue entre el núcleo del sistema operativo (SunOs) y el entorno operativo en general (Solaris). Por otro lado, hasta la actualidad Solaris es un sistema certificado oficialmente como versión de Unix. Soporta arquitecturas SPARC y x86 para servidores y estaciones de trabajo, aunque algunas versiones anteriores soportaba también la arquitectura PowePC.

Solaris mantiene una gran reputación de obtener un buen rendimiento en entornos donde prima el multiprocesamiento simétrico gracias a que soporta y gestiona un gran número de CPUs. También incluye soporte para aplicaciones de 64 bits SPARC desde Solaris 7.

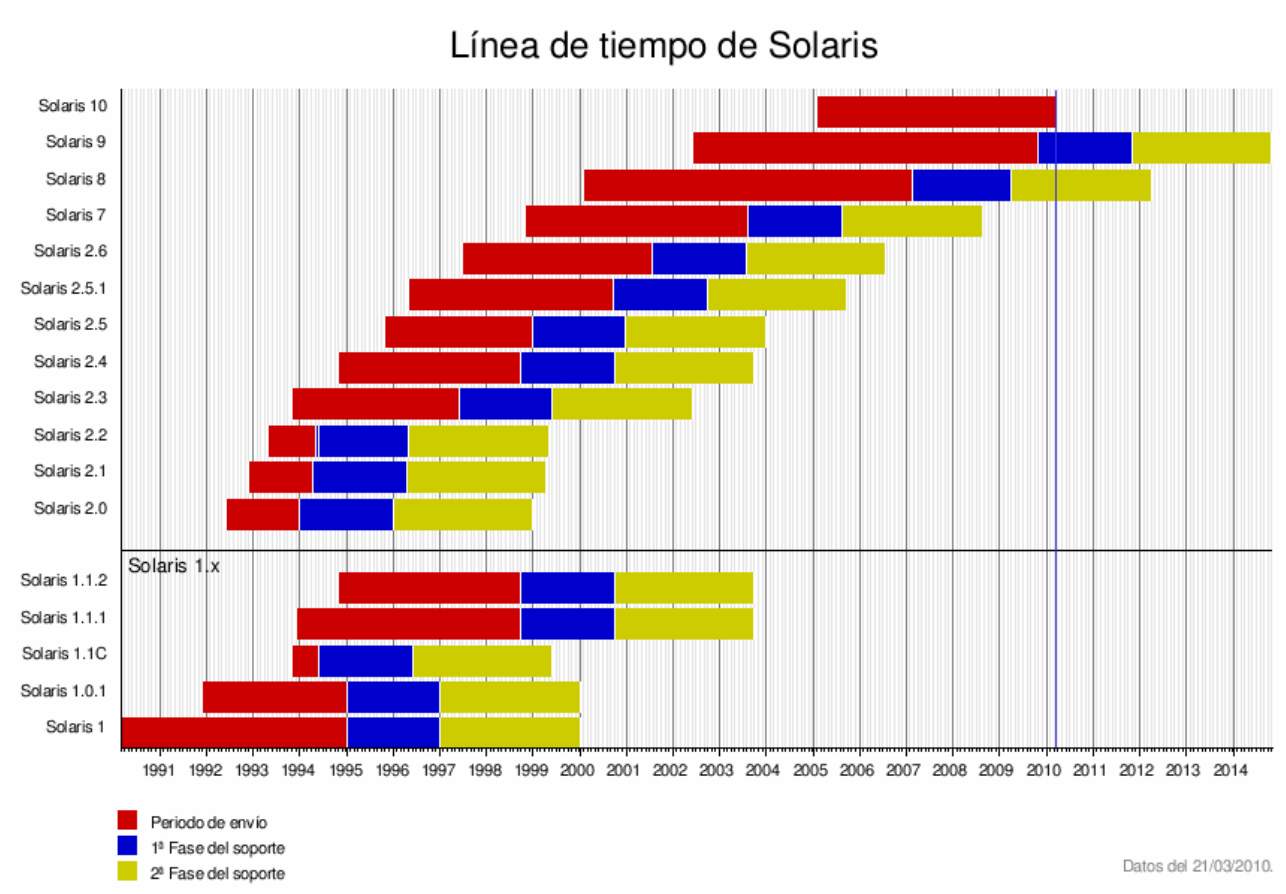
A continuación mostraremos un resumen de las versiones de Solaris así como de sus características más destacables:

- ✔ Solaris 2.0 que incluía SunOS 5.0 fue lanzada en junio de 1992, fue la primera versión preliminar.
- ✔ Solaris 2.1 con SunOS 5.1 lanzada en diciembre de 1992 fue la primera versión para Solaris x86.
- ✔ Solaris 2.2 con SunOS 5.2 lanzada en mayo de 1993 con exclusividad para arquitecturas SPARC.
- ✔ Solaris 2.3 con SunOS 5.3 lanzada en noviembre de 1993.
- ✔ Solaris 2.4 con SunOS 5.4 lanzada en noviembre de 1994, primera versión unificada SPARC/x86.
- ✔ Solaris 2.5 con SunOS 5.5 lanzada en noviembre de 1995, primera versión en soportar UltraSPARC e incluye NFSv3 y NFS/TCP.
- ✔ Solaris 2.5.1 con SunOS 5.5.1 lanzada en mayo de 1996, fue la primera y única versión que soportó la plataforma PowerPC.
- ✔ Solaris 2.6 con SunOS 5.6 lanzada en julio de 1997, incluía soporte para grandes archivos.
- ✔ Solaris 7 con SunOS 5.7 lanzada en noviembre de 1998, fue la primera versión de 64

bits para la plataforma UltraSPARC.

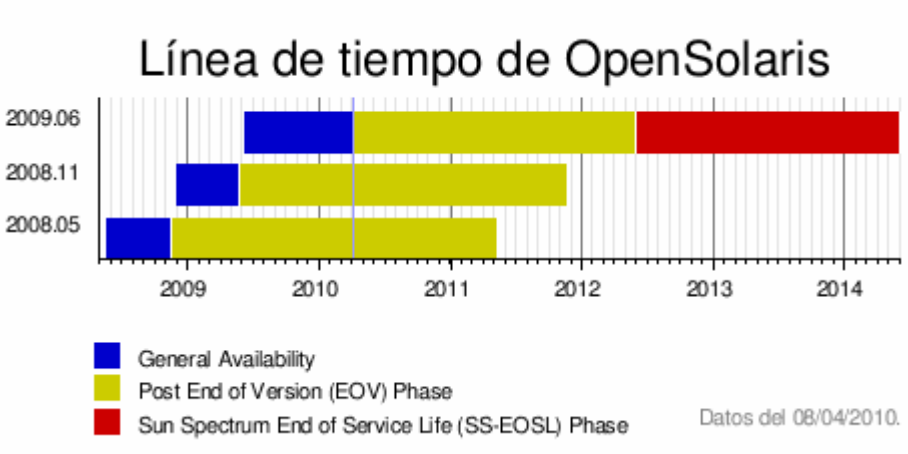
- ✔ Solaris 8 con SunOS 5.8 lanzado en febrero de 2000 incluye Multipath I/O, IPv6 y Ipsec.
- ✔ Solaris 9 con SunOS 5.9 lanzado en mayo de 2002 para SPARC y en enero de 2003 para x86. Esta versión incluye Solaris Volumen Manager además de añadir compatibilidad con Linux.
- ✔ Solaris 10 con SunOS 5.10 lanzado en enero de 2005 incluye soporte para plataformas AMD64/EM64T, además de numerosas mejoras respecto a la anterior, pero las más destacadas son: Dtrace, Solaris Containers, Service Management Facility (SMF) para reemplazar al sistema de init.d, NFSv4, GRUB (como cargador de arranque para plataformas x86), soporta iSCSI y un nuevo sistema de ficheros, ZFS.

A continuación se muestra un gráfico cronológico de las versión indicadas anteriormente:



Por otro lado, tenemos Opensolaris. Es un sistema operativo libre (con licencia CDDL de tipo copyleft) publicado en 2005 a partir de Solaris. Se deriva de código base del Unix System V, aunque gran parte de él ha sido modificado desde las versiones de inicios de la década los 1990. Es la única versión de System V disponible en código abierto. Alrededor de 16.400 miembros están registrados en la comunidad de OpenSolaris hacen de esta distribución una de las más apoyadas por la comunidad de desarrolladores [14].

- ✔ OpenSolaris 2008.05 fue lanzad en mayo de 2008 para arquitecturas i386, e incluía el entorno de escritorio GNOME, el suite de ofimática OpenOffice y el sistemas de ficheros ZFS.
- ✔ OpenSolaris 2008.11 fue lanzada en noviembre de 2008 con bastante éxito, sobretodo la versión para entornos de escritorio pues que se convertía en una de las distribuciones de código libre más "amigables" hasta la fecha.
- ✔ OpenSolaris 2009.06 en junio de 2006 y es la primera versión que incluye la plataforma SPARC.



4.3.2 Runlevels

Las distribuciones basadas en Solaris, como OpenSolaris utilizan un sistema de arranque similar a los empleados en Linux. Contiene ocho niveles y a continuación se muestra un pequeño resumen de ellos:

- 1, s y S – Modo monousuario y administración. Arranca exclusivamente los servicios imprescindibles para el correcto funcionamiento del kernel y solo se montan los filesystems básicos de sistema.
- 2 – Modo multiusuario, sin servicios de red.
- 3 – Modo multiusuario, con servicios de red.
- 4 – No usado, normalmente se utiliza como runlevel personalizable, para uso del administrador.
- 5 – Runlevel de apagado.
- 6 – Runlevel de reinicio. Para todos los servicio y reinicia el servidor en el modo por defecto (runlevel 3).

4.3.3 Zonas

Introducción

Las zonas son una característica de Solaris que permite virtualizar servicios del sistema operativo, y proporcionar un entorno aislado y seguro para la ejecución de aplicaciones. Además, las zonas proporcionan un nivel de abstracción que separa las aplicaciones de los atributos físicos del hardware. Este aislamiento evita que los procesos que se están ejecutando en una zona sean controlados o se vean afectados por los procesos que se están ejecutando en otras zonas. Incluso un proceso que se está ejecutando con credenciales de superusuario no puede ver ni afectar a la actividad que se esté realizando en otras zonas [15].

Por otro lado, son idóneas para entornos que consolidan varias aplicaciones en un único servidor. Las zonas permiten la reasignación dinámica de recursos del sistema, es decir, podemos mover recursos entra las zonas según la carga del sistema lo precise.

Por último, indicar que un proceso asignado a una zona puede manipular, supervisar y comunicarse directamente con otros procesos asignados a la misma zona, pero no puede llevar a cabo estas funciones con procesos que están asignados a otras zonas del sistema o con procesos que no están asignados a ninguna zona. Los procesos asignados a diferentes zonas sólo pueden comunicarse a través de las API de red.

Tipos

Existen dos tipos de zonas:

- Global: Por defecto, el sistema contiene una de estas zonas desde la cual se puede administrar el sistema y las zonas no globales. También es la única zona desde donde se puede configurar, instalar, desinstalar y gestionar una zona no global.
- No global: entorno del sistema donde los procesos que se ejecutan en esta zona están aislados del resto de sistema.

Estados

Cada zona tiene un estado que permite saber como se encuentra cada zona en todo momento. Los posibles estados en los que puede estar una zona son los seis siguientes:

- Configurada: La configuración de la zona está completa, es correcta y se envía a una ubicación de almacenamiento estable. En este estado, la zona no puede ser arrancada.
- Incompleta: Mientras dura el proceso de instalación o desinstalación, el sistema asigna el estado incompleta a la zonas en progreso.
- Instalada: La configuración de la zona se instancia en el sistema, pero la zona no tiene ninguna plataforma virtual asociada. En este estado la zona ya puede ser arrancada.
- Lista: Estado previa al de ejecución, en el cual se establece la plataforma virtual para la zona, se configuran los interfaces de red, se montan los sistemas de ficheros y los dispositivos se configuran.
- Ejecutándose: Se pasa a este estado cuando en una zona se está ejecutando procesos del usuario asociados a la misma.
- Cerrada o cerrándose: Se trata de estados de transición visibles cuando se está deteniendo la zona.

Comandos

Con los siguientes comandos podemos realizar la mayoría de acciones necesarios para utilizar/administrar una zona:

- zonecfg: Se utiliza para crear, eliminar y modificar las propiedades de las zonas.
- zoneadm: Utilizado para administrar las zonas. Se pueden arrancar, parar, reiniciar, instalar, desinstalar, clonar, mover, etc...
- zlogin: Sirve para logarse en la zona.

Otro comando de muy útil es zonename que retorna el nombre de la zona donde te encuentras actualmente.

Definición y configuración

Existen numerosas propiedades que podemos configurar en nuestras zonas para darles la una configuración lo más ajustada posible a nuestros objetivos, tal es la magnitud que no podemos numerarlas aquí todas, pero los puntos más importantes a la hora de crear y configurar una zona son los siguientes:

1. Propiedades básicas:
   a) Arranque automático: Configurando la propiedad *autoboot* a true determinamos que la zona arranque cuando arranca lo zona global, que por defecto es la zona del sistema.
   b) La propiedad *zonepath* es la ruta del directorio root de la zona y es relativo al directorio root de la zona global o, en su defecto, debe colgar de un directorio con permisos exclusivamente de root.
2. Recursos de almacenamiento:
   a) Para determinar cómo y dónde se montan los sistemas de ficheros utilizaremos el parámetro *fs* y las variables siguientes:
      - *dir*: Indicaremos el punto de montaje para el sistema de archivos.
      - *special*: Nombre del dispositivo o directorio a partir de la zona global que montaremos en nuestra zona.
      - *type*: Tipo de filesystem que vamos a añadir.

      La mejor opción para aprovechar al máximo la conjunción entre el sistema de ficheros ZFS y las zonas es asignar un filesystem ZFS a nuestra zona del siguiente modo:

```
# zonecfg -z zprueba
zonecfg:zprueba> add fs
zonecfg:zprueba:fs> set type=zfs
zonecfg:zprueba:fs> set special=rpool/zona/zprueba
zonecfg:zprueba:fs> set dir=/export/shared
zonecfg:zprueba:fs> end
```

   b) Con *dataset* definiremos un filesystem como medio de almacenamiento a nuestra zona que se visualizará y montará en la zona no global y dejará de estar visible en la zona global:

```
zonecfg:zprueba> add dataset
zonecfg:zprueba> set name=rpool/datos
zonecfg:zprueba> end
```

      si el *dateaset* se encuentra en ZFS, el comando zoneadm install crea automáticamente un sistema de archivos ZFS (conjunto de datos) para el filesystem cuando se instala la zona .

3. Recursos de red: Podemos configurar el acceso de nuestra zona a la red de múltiples modos:

    a) Podemos declarar una IP exclusiva para nuestra zona, además deberemos indicarle la interfaz por donde accederemos a la red:

```
zonecfg:zprueba> set ip-type=exclusive
zonecfg:zprueba> add net
zonecfg:zprueba:net> set physical=rge32001
zonecfg:zprueba:net> end
```

    b) Otra opción es la de tener una IP compartida, de este modo, definimos la ip y el interfaz:

```
zonecfg:zprueba> add net
zonecfg:zprueba:net> set physical=rge0
zonecfg:zprueba:net> set address=192.168.1.100
zonecfg:zprueba:net> end
```

4. Recursos de procesamiento.
    a) CPU: Para determinar el uso el modo de uso de las CPU's de nuestro sistema tenemos varias propiedades:

        - ***capped-cpu***: permite dar visibilidad a la zona sobre las CPU's de la zona global de sistema y podemos asignar a cada zona el número de CPU's que puede utilizar, asignando un valor en la propiedad ***ncpus*** de la zona. Con un 1 asignamos el 100% de una de las CPU's.

```
zonecfg:zprueba> add capped-cpu
zonecfg:zprueba:dedicated-cpu> set ncpus=2,5
zonecfg:zprueba:dedicated-cpu> end
```

        - ***dedicated-cpu***: con esta propiedad damos la exclusividad de uso de las CPU's asignadas a la zona, con lo cual, ninguna otra zona podrá hacer uso de estas CPU's reservadas y podemos asignar a cada zona el número de CPU's que puede utilizar, asignando un valor en la propiedad *ncpus* de la zona. Con un 1 asignamos el 100% de una de las CPU's, pero también podemos asignar un rango. A continuación, se muestra un ejemplo:

```
zonecfg:zprueba> add dedicated-cpu
zonecfg:zprueba:dedicated-cpu> set ncpus=1-5
zonecfg:zprueba:dedicated-cpu> set importance=2
zonecfg:zprueba:dedicated-cpu> end
```

        - Para asegurar un reparto equitativo del uso de CPU entre la zonas

existo otra propiedad a tener en cuenta ***shedulilng-class***, si le asignamos el modo FSS (programador de reparto justo) el sistema se basará en las cargas de trabajo de cada zona. Esa importancia se asigna indicando un valor a la variable ***cpu-shares***.

```
zonecfg:zprueba> set scheduling-class=FSS
zonecfg:zprueba> set cpu-shares=80
```

b) Memoria: para establecer limites en cuanto al uso de la memoria del sistema por parte de las zonas, podemos asignar cualquiera de los siguientes valores a la propiedad *capped-memory* de cada zona (y al menos una de ellas):

- Le daremos el valor *physical* si lo que limitar la memoria para la zona.
- Con le valor *locked* bloqueamos cierto espacio de memoria para el uso de nuestra zona.
- Y por último, podemos optar por asignar el valor a la variable *swap* para definir un tamaño de swap para nuestra zona.

El siguiente ejemplo ilustra las opciones anteriormente descritas:

```
zonecfg:zprueba> add capped-memory
zonecfg:zprueba:capped-memory> set physical=100m
zonecfg:zprueba:capped-memory> set swap=200m
zonecfg:zprueba:capped-memory> set locked=40m
zonecfg:zprueba:capped-memory> end
```

c) Existen muchas otras características para personalizar y adaptar a nuestras necesitadas cualquiera de nuestras zonas. A continuación mostramos algunas otras dignas de mencionar:

- *max-lwps*: con este recurso determinamos el número máximo de procesos ligeros (LWP) que puede ejecutar simultáneamente una zona determinada, con esto mejoramos el aislamiento del recurso al evitar que dichos procesos afecten al rendimiento de otras zonas.
- *max-msg-ids*: Número máximo de identificadores de cola de mensajes permitidos para esta zona.
- *max-sem-ids*: Número máximo de identificadores de semáforos permitidos para esta zona.
- *max-shm-ids*: Número máximo de identificadores de memoria compartida permitidos para esta zona.
- *max-shm-memory*: Cantidad total de memoria compartida System V permitida para esta zona, en bytes.

5. Verificamos, instalamos y arrancar la zona.

zonecfg -z nombre_zona verify

zonecfg -z nombre_zona commit

zoneadm -z nombre_zona verify

zoneadm -z nombre_zona install

zoneadm -z nombre_zona boot

Otra opción ha tener en cuenta es el backup de la configuración de nuestras zonas. Podemos el comando realizar *"zonecfg -z my-zone export > my-zone.config"* para realizar un volcado de la configuración de nuestra zona, ya sea para destinarlo a backup, para migrarla o para duplicarla en otro sistema.

El mejor modo de explicar la creación y administración de las zonas en con un ejemplo. Por eso, he preparado el siguiente ejemplo.

Supongamos que tenemos un sistema donde queremos ejecutar diversas aplicaciones, pero queremos que estén lo más aisladas posibles entre si. Además, debido a las condiciones contractuales de los servicios debemos asignar un número determinado de recursos. Para ello utilizaremos nuestro sistema OpenSolaris complementado con las zonas para abarcar todos los requisitos del proyecto.

El primer paso será definir cuantas zonas deseamos crear y la configuración que tendrán cada una de ellas. Tenemos un cliente, por ello crearemos una zona no global a las que llamaremos zcliente1. Asignaremos una cantidad de cpu, memoria y otros parámetros para limitar la zona, además de asignar un filesystem donde es almacenaran todos los ficheros necesarios para la ejecución de los procesos de los servicios de nuestros clientes.

1. Un paso previo a la creación de la cola es la creación de una filesystem zfs que luego utilizaremos pasa asignarle al zonepath y como almacenamiento exclusivo para nuestra zona:

```
root@opensolaris:~# zfs create rpool/zonas
root@opensolaris:~# zfs create rpool/zcliente1
root@opensolaris:~# zfs set quota=20GB rpool/zcliente1
root@opensolaris:~# mkdir /rpool/zcliente1/usr_local
root@opensolaris:~# chmod 700 /rpool/zcliente1/
root@opensolaris:~# chmod 775 /rpool/zcliente1/usr_local/
```

2. El siguiente paso de todos es logarnos a un terminal con root y creamos las zonas:

```
root@opensolaris:~# zonecfg -z zcliente1
zcliente1: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zcliente1> create
```

3. A continuación, definiremos las variables imprescindibles para el funcionamiento de nuestra zona. Al zonepath le asignaremos el filesystem creado en el primer paso y definiremos que la zona arranque junto con el sistema tras un reinicio:

```
zonecfg:zcliente1> set zonepath=/rpool/zcliente1
zonecfg:zcliente1> set autoboot=true
```

4. Llegados a este punto, ya podemos añadir a nuestra zona los

```
zonecfg:zcliente1> add fs
zonecfg:zcliente1:fs> set dir=/usr/local
zonecfg:zcliente1:fs> set special=/rpool/zcliente1/usr_local
zonecfg:zcliente1:fs> set type=lofs
zonecfg:zcliente1:fs> end
```

5. Una vez finalizado en tema del almacenamiento pasamos a determinar los recursos que permitiremos utilizar a nuestras zona, tanto a nivel de CPU como de memoria:

```
zonecfg:zcliente1> add capped-cpu
zonecfg:zcliente1:capped-cpu> set ncpus=1,75
zonecfg:zcliente1:capped-cpu> end
zonecfg:zcliente1> set scheduling-class=FSS
zonecfg:zcliente1> add capped-memory
zonecfg:zcliente1:capped-memory> set physical=100m
zonecfg:zcliente1:capped-memory> set swap=200m
zonecfg:zcliente1:capped-memory> set locked=25m
zonecfg:zcliente1:capped-memory> end
```

6. Por último, confirmaremos los cambios realizados en la zona y verificaremos que no tengan fallos de sintaxis:

```
zonecfg:zcliente1> commit
zonecfg:zcliente1> verify
zonecfg:zcliente1> exit
```

7. Tras realizar satisfactoriamente los pasos anteriores procedemos a instalar la zona,el sistema instalará algunas paquetes necesarios para la buena ejecución del administrador de zonas y, si no se produce ningún error, tendremos nuestra zona instalada:

```
root@opensolaris:~# zoneadm -z zcliente1 install
Publisher: Using opensolaris.org
(http://pkg.opensolaris.org/release/)
       Image: Preparing at /rpool/zcliente1/root.
       Cache: Using /var/pkg/download.
Sanity Check: Looking for 'entire' incorporation.
```

```
  Installing: Core System (output follows)
DOWNLOAD                                PKGS       FILES     XFER (MB)
Completed                             20/20   3021/3021   42.55/42.55

PHASE                                          ACTIONS
Install Phase                                 5747/5747
  Installing: Additional Packages (output follows)
DOWNLOAD                          PKGS       FILES     XFER (MB)
Completed                         37/37   5600/5600   32.53/32.53

PHASE                                          ACTIONS
Install Phase                                 7338/7338

        Note: Man pages can be obtained by installing SUNWman
 Postinstall: Copying SMF seed repository ... done.
 Postinstall: Applying workarounds.
        Done: Installation completed in 948,033 seconds.

  Next Steps: Boot the zone, then log into the zone console
              (zlogin -C) to complete the configuration process
```

8. Llegados a este punto, ya podemos logarnos en nuestra zona y lanzar las aplicaciones que deseemos dentro de nuestra zona e independientemente del resto del sistema:

```
root@opensolaris:~# zlogin zcliente1
[Connected to zone 'zcliente1' pts/3]
Sun Microsystems Inc.   SunOS 5.11  snv_111b    November 2008
root@zcliente1:~#
```

4.3.4 Auto recuperación preventiva

La "Tecnología preventiva de auto recuperación" (PSH, Predictive Self-Healing) es una tecnología desarrollada por Sun Microsystems que se utilizan en el núcleo de los sistemas operativos Solaris que reduce los riesgos y aumenta la disponibilidad del equipo, además permite diagnosticar, aislar y recuperar las fallas existentes en los dispositivos de E/S o memoria [16].

4.3.5 Dynamic Tracing (Dtrace)

Denominado también Dtrace fue diseñado e implementado por Bryan Cantrill, Mike Shapiro y Adam Leventhal de Sun Microsystems, es un framework de rastreo y monitorización dinámica, busca llegar a la raíz de los problemas de rendimiento en tiempo real, más concretamente, diagnostica problemas de kernel y aplicaciones en sistemas de producción en tiempo real. Esta herramienta trabaja utilizando sondas inteligentes del sistema que pueden acceder a áreas de más lento rendimiento o con cuellos de botella, y todo ello sin coste alguna para el rendimiento del sistema, ya que es una herramienta no intrusiva. Además, permiten visualizar mejor la actividad del núcleo y de las aplicaciones que corren en el sistema. Originalmente desarrollado para Solaris, ha sido lanado bajo licencia CDDL (Licencia Común de Desarrollo y Distribución) y ha sido portado a otros sistemas operativos tipo Unix [17].

Los scripts utilizados en Dtrace están escritos en lenguaje de programación D. Este lenguaje es un subconjunto del lenguaje C, con funciones y variables de rastreo añadidas.

4.4 Comparativa

Tras un numerosas horas de investigación sobre cada uno de los sistemas podemos extraer las siguientes ventaja e inconvenientes de cada unos de los sistemas operativos, Debian y Solaris.

Para empezar tenemos que Debian tiene grandes ventajas como:

- ✔ Portabilidad: es capaz de ejecutarse en numerosas arquitecturas.
- ✔ Código abierto: se puede modificar al gusto y optimizar al máximo.
- ✔ Gran comunidad de desarrollo.
- ✔ Distribuciones: existen una gran variedad de distribuciones que derivan de Debian y que se utilizan en diversos entornos.
- ✔ Estabilidad: es uno de los sistemas más estables del mercado. Gran parte de la culpa de esto es del kernel que alberga Debian.
- ✔ Modularidad: gracias al sistema de módulos del kernel, podemos dejar arrancados los módulos estrictamente necesarios, de este modo, aumentamos el rendimiento general del sistema.
- ✔ LVM: sistema de almacenamiento con gran facilidad de administración.

En su contra y como podremos ver más adelante, tiene lo siguientes inconvenientes

- ✗ Por defecto, se obtiene un rendimiento inferior en comparación con otros sistemas.
- ✗ Difícil implantación en sistemas productivos críticos por su tipología de soporte (aunque existen otras distribuciones con soporte de empresas externas con gran implantación en sistemas productivos).
- ✗ Plataforma de virtualización: Actualmente, casi todos los sistemas operativos tienen su propia plataforma interna donde podemos crear particiones lógicas de nuestro sistema para virtualizar, ya sea otros sistemas o servicios, de modo que se aumenta el aislamiento del resto del sistema.

En cuanto a OpenSolaris, tenemos las siguientes ventajas:

- ✔ Diseño orientado a entornos con gran criticidad y de alto rendimiento.
- ✔ Sistema de almacenamiento ZFS nativo.
- ✔ Soporte: la versión de código propietario Solaris tiene un gran soporte lo que hace que sea un sistema idóneo para sistemas productivos críticos.
- ✔ Sistema de Zonas: sistema nativo de virtualización que nos provee de gran un aislamiento los servicios albergados en nuestro sistema.
- ✔ Dtrace: como ya hemos hablado anteriormente, se trata de una gran herramienta para detectar cuellos de botella en el rendimiento de nuestro sistema.

Y, como no, los inconvenientes:

- ✗ Portabilidad limitada (SPARC y ….)
- ✗ Limitación en cuanto al acceso a diversas partes del código ya que, gran parte de él, continua siendo código propietario.

5. Sistemas de almacenamiento

Una parte vital de nuestros sistemas son su capacidad de almacenamiento. Como ya sabemos los datos dentro del hardware son volátiles, por eso necesitamos un sistema físico de almacenamiento.

Llegados a este punto, debemos utilizar un gestor que nos facilite la gestión de estos recursos, así como la distribución de los mismos, es decir, una capa lógica entre nuestros dispositivos físicos (véase discos) y el administrador del sistema.

Existen numerosos, pero para los sistemas usados en este proyecto los más utilizados son LVM para Linux y ZFS para Solaris.

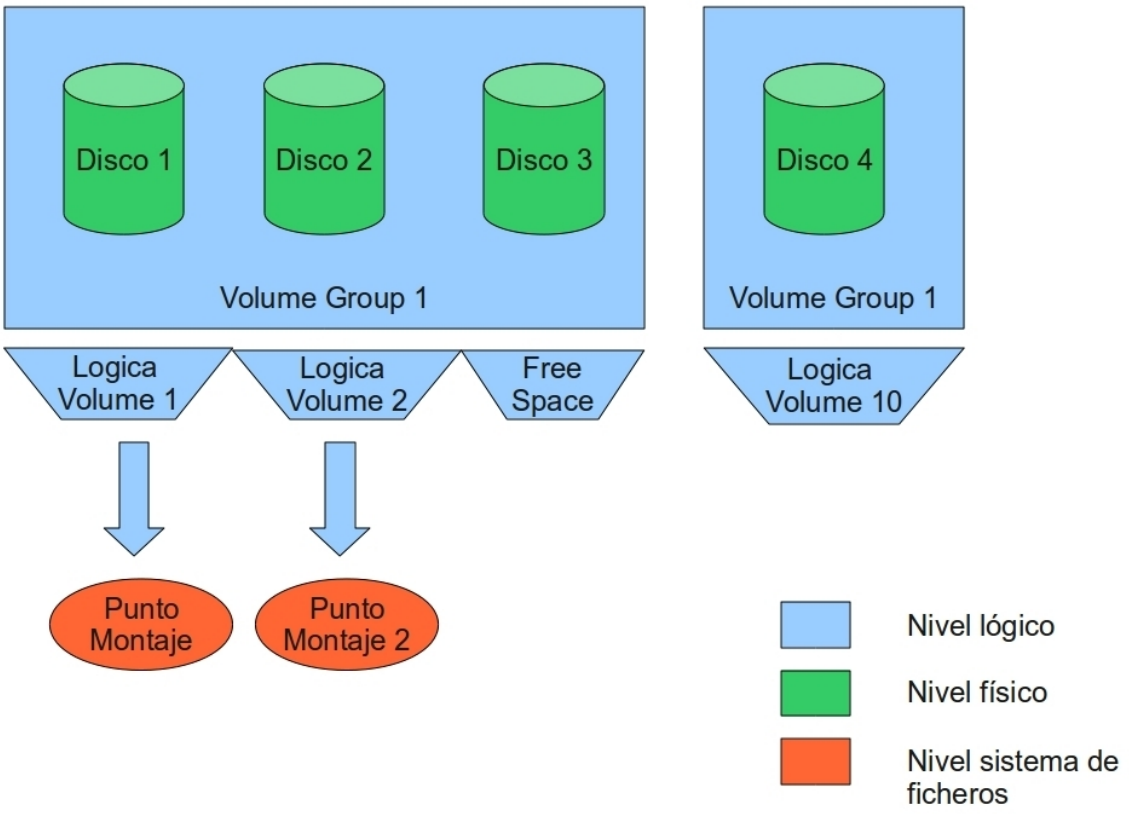
A continuación, describiremos cada uno de ellos.

5.1 LVM (Logical Volume Manager)

LVM es una implementación de un administrador de volúmenes lógicos para el kernel de Linux. Se escribió originalmente en 1998 por Heinz Mauelshagem, que se basó en el administrador de volúmenes de Veritas usado en sistemas HP-UX [18].

Las partes lógicas que componen el LVM son las siguientes:

a) Volúmenes físicos (PV): Son los discos físicos o particiones de los mismos.
b) Grupos de volúmenes (VG): Son una unidad lógica que engloba un número determinado de volúmenes físicos y lógicos, es decir, son el nexo de unión entre los volúmenes físicos y lógicos. Otra característica a destacar es la posibilidad de asignar un nombre a los grupos de volúmenes, lo que nos proporciona la oportunidad de utilizar este atributo para indicar las características de los datos que alojará, p.e. facturasvg, aplicacionesvg o bbddvg.
c) Volúmenes lógicos (LV): Son unas estructuras dentro de los grupos de volúmenes con unas dimensiones determinadas y ampliables con las características equivalentes a las de una partición de un disco.



El LVM ofrece multitud de características, pero las más importantes de LVM son:

- Redimensionados "en caliente" de volúmenes lógicos sin necesidad de desmontar el punto de montaje.
- Inserción/eliminación/reasignación de volúmenes físicos en los grupos de volúmenes.

Comandos:

Existen numerosos comandos para manipular los volúmenes físicos, lógicos y grupos de volúmenes. Se dividen en tres grandes grupos: informativos, de creación y de modificación/eliminación. A continuación se describen algunos de los más utilizados e imprescindibles para nuestro próximo ejemplo [19]:

- Informativos:
  - *vgdisplay*: Este comando nos muestra toda la información de un grupo de volúmenes o en caso de omitir el nombre del grupo, nos lista todos los existentes.
  - *lvdisplay*: Este comando nos permite visualizar volúmenes lógicos dentro de un grupo de volúmenes determinado.
  - df: Además de permitirnos saber el uso de nuestros filesystems, nos permite saber que volumen lógico está relacionado con nuestro punto de montaje.

- Creación:
  - *vgcreate*: Para crear un nuevo grupo de volúmenes utilizaremos este comando y como parámetro le indicaremos en nombre y, si lo deseamos, podemos añadirle algún disco. La sintaxis es la siguiente:

    ```
    vgcreate VolumeGroupName PhysicalVolumePath[PhysicalVolumePath..]
    ```

  - *lvcreate*: Con este comando crearemos un volumen lógico un en grupo de volúmenes existente. Este comando tiene numerosos parámetros, pero los más importantes son:

    - -L *tamaño* : con este parámetro le indicamos el tamaño que le deseamos dar el volumen lógico, ya sea en kilobytes (K), megabytes (M), etc.
    - -n *nombre* : con este parámetro le indicamos el nombre de nuestro volumen lógico.

    Y la sintaxis es:

    ```
    lvcreate    [ -L/--size LogicalVolumeSize[kKmMgGtT]
                [-n/--name  LogicalVolumeName]
                VolumeGroup- Name [PhysicalVolumePath...]
    ```

  - *mkfs.ext3*: Con este comando, o sus variantes ext2, ext4, etc., damos un formato determinado a nuestro volumen lógico, acción necesaria para poder utilizarlo, a excepción de algunos sistemas de bases de datos que se les da un volumen lógico sin formato para que ellos mismos optimicen el rendimiento con un formato propio.

- Modificación:
  - *lvextend*: Este comando resulta de los más versátiles de los que dispones LVM. Con él podemos ampliar la capacidad de nuestro volumen lógico siempre y cuando tengamos el mismo espacio libre en el grupo de volúmenes al que pertenece. Además, podremos reducirlo si indicamos en el parámetro "size" un tamaño menor que el actual aunque, normalmente, implica pérdida de datos. Por este motivo es recomendable hacer un backup previo de los ficheros.

    A continuación, mostramos una versión reducida de los posibles parámetros de este comando:

    ```
    lvextend {-l/--extents  [+]LogicalExtentsNumber[%{VG|LV|PVS|
    FREE}]  |  -L/--size [+]LogicalVolumeSize[kKmMgGtT]} LogicalVol-
           umePath [PhysicalVolumePath...]
    ```

  - *resize2fs*: Utilizaremos este comando para refrescar nuestro filesystem y para reconocer la ampliación que habremos realizado con un lvextend. A

continuación, mostramos la sintaxis:

```
resize2fs [-fFpPM] device [ size ]
```

- *vgextend*: Para asignar disco a un grupo de volúmenes ya existente utilizaremos este comando. El volumen físico que signemos no debe pertenecer a otro grupo de volúmenes. La sintaxis es la siguiente:

```
vgextend VolumeGroupName   PhysicalDevicePath
```

- *vgreduce*: Para eliminar un disco de un grupo de volúmenes utilizaremos este comando. Para poder eliminar el disco no debe de haber ningún volumen lógico utilizando este disco sino, implicaría pérdida de datos. La sintaxis es la siguiente:

```
vgreduce VolumeGroupName  [PhysicalVolumePath...]
```

Pasos a seguir para implementar un LVM:

1. Análisis: Definir las necesidades para el entorno que queremos crear y diseñar los grupos de volúmenes acorde a ellos. Por ejemplo, deseamos tener en nuestro sistema varios servidores de aplicaciones web (p.e. un Tomcat y un Jboss), además una base de datos, posiblemente PostgreSQL.

2. Diseño: Crearemos dos grupos de volúmenes: uno para aplicaciones y otro para base de datos.

a) Un grupo de volúmenes para aplicaciones, llamado appvg, donde se crearán todos los volúmenes lógicos necesarios para aplicaciones, en nuestro ejemplo, crearemos dos volúmenes lógicos, para un futuro Tomcat y Jboss, llamados tomcatlv y jbosslv, respectivamente.

b) Otro grupo de volúmenes para bases de datos, llamado bddvg, donde crearemos todos los volúmenes necesarios para almacenar nuestras bases de datos. En nuestro ejemplo, crearemos un volumen lógico llamado postgreslv.

3. Ejecución:

a) Creamos los volúmenes físicos:

- Para ello primero revisamos los discos que tenemos disponibles:

```
administrador@debian:~$ ls -l /dev/sd*
brw-rw---- 1 root disk 8,  0 abr  4 16:20 /dev/sda
brw-rw---- 1 root disk 8,  1 abr  4 16:20 /dev/sda1
brw-rw---- 1 root disk 8,  2 abr  4 16:20 /dev/sda2
brw-rw---- 1 root disk 8, 16 abr  4 16:20 /dev/sdb
brw-rw---- 1 root disk 8, 32 abr  4 16:20 /dev/sdc
```

Con esto podemos ver los discos que tenemos: sda, sdb y sdc.
De estos tres observamos que el sda está siendo utilizado mediante un "pvdisplay":

```
debian:~# pvdisplay
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               debian
  PV Size               9,76 GB / not usable 1,56 MB
  Allocatable           yes (but full)
  PE Size (KByte)       4096
  Total PE              2498
  Free PE               0
  Allocated PE          2498
  PV UUID               Kjomqo-NLWa-9VN4-QOZS-O8AP-zCof-Urh9q2
```

- Ahora crearemos los volúmenes físicos con los discos que tenemos disponibles, mediante el comando *pvcreate*:

```
debian:~# pvcreate /dev/sdb
  Physical volume "/dev/sdb" successfully created
debian:~# pvcreate /dev/sdc
  Physical volume "/dev/sdc" successfully created
```

Y si repetimos el paso anterior, podemos verlos:

```
 "/dev/sdb" is a new physical volume of "5,00 GB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb
  VG Name
  PV Size               5,00 GB
  Allocatable           NO
  PE Size (KByte)       0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               o2tLvt-4SnX-hmQg-0OYs-DIbx-rJ9u-d2epfM

  "/dev/sdc" is a new physical volume of "5,00 GB"
  --- NEW Physical volume ---
  PV Name               /dev/sdc
  VG Name
  PV Size               5,00 GB
  Allocatable           NO
  PE Size (KByte)       0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               dCyk4a-dYxt-SQan-TL4m-YZXU-g1Dv-j023Da
```

b) Una vez tenemos los discos creados, pasamos a crear los grupos de volúmenes. Para ello, utilizaremos el comando vgcreate:

```
debian:~# vgcreate bddvg /dev/sdb
  Volume group "bddvg" successfully created
debian:~# vgcreate appvg /dev/sdc
  Volume group "appvg" successfully created
```

c) Una vez definidos los grupos de volúmenes, ya podemos crear los volúmenes lógicos que deseemos mientras tengamos espacio suficiente en el grupo (en caso de falta de espacio podemos aumentar los volúmenes físicos del grupo para ganar espacio). Para crear el volumen lógico del Tomcat realizaremos el siguiente comando:

```
debian:~# lvcreate -L 1G -n tomcatlv appvg
  Logical volume "tomcatlv" created
```

Si queremos observar la información del volumen lógico que acabamos de crear utilizaríamos el siguiente comando:

```
debian:~# lvdisplay /dev/appvg/tomcatlv
  --- Logical volume ---
  LV Name                /dev/appvg/tomcatlv
  VG Name                appvg
  LV UUID                sGOjIp-h8oc-zOG6-Qyvf-8ybI-zYqN-oPoTX
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                1,00 GB
  Current LE             256
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:2
```

Del mismo modo que hemos creado éste, podemos crear el resto:

```
debian:~# lvcreate -L 1G -n jbosslv appvg
  Logical volume "jbosslv" created
debian:~# lvcreate -L 1G -n postgreslv bddvg
  Logical volume "postgreslv" created
```

Después de crear los volúmenes lógicos, debemos darles formato con el comando mke2fs del siguiente modo:

```
debian:~# mkfs.ext3 /dev/appvg/tomcatlv
mke2fs 1.41.3 (12-Oct-2008)
Etiqueta del sistema de ficheros=
Tipo de SO: Linux
Tamaño del bloque=4096 (bitácora=2)
Tamaño del fragmento=4096 (bitácora=2)
65536 nodos-i, 262144 bloques
13107 bloques (5.00%) reservados para el superusuario
Primer bloque de datos=0
Número máximo de bloques del sistema de ficheros=268435456
8 bloque de grupos
32768 bloques por grupo, 32768 fragmentos por grupo
8192 nodos-i por grupo
Respaldo del superbloque guardado en los bloques:
   32768, 98304, 163840, 229376

Escribiendo las tablas de nodos-i: hecho
```

```
      Creating journal (8192 blocks): hecho
      Escribiendo superbloques y la información contable del sistema de
ficheros: hecho

      Este sistema de ficheros se revisará automáticamente cada 22
montajes o
      180 días, lo que suceda primero.  Utilice tune2fs -c o -i para
cambiarlo.
      debian:~#
```

d) El siguiente y último paso es montar los volúmenes lógicos creados en la ubicación que deseemos dentro de nuestra jerarquía de ficheros. Para realizarlo seguiremos los siguiente pasos:

- Creamos un punto de montaje con un "`mkdir /nombre_directorio`"

```
debian:~# mkdir /app/tomcat
```

- Montamos el volumen lógico con el un "`mount nombre_lvol nombre_directorio`"

```
debian:~# mount /dev/appvg/tomcatlv /app/tomcat
```

Tras el montaje del nuevo filesystem, podemos apreciarlo mediante el comando "df":

```
debian:~# df -h /app/tomcat
S.ficheros          Tamaño Usado  Disp Uso% Montado en
/dev/mapper/appvg-tomcatlv
                      1008M   34M  924M   4% /app/tomcat
```

- El siguiente paso seria cambiar permisos del directorio al usuarios y grupo propietarios del mismo, p.e. "`chown utomcat:gtomcat /app/tomcat`"
- El último paso es añadir al fichero /etc/fstab una entrada con el nuevo filesystem que hemos creado. Con ello, conseguimos que el propio sistema monte el filesystem que acabamos de crear al arrancar el sistema, por ejemplo tras un reinicio.

Todos estos pasos pueden ser automatizados mediante un script, con la consecuente ganancia en tiempo. Incluso utilizar el stdin y stdout para importar lo datos de un fichero y redirigir la salida a otro fichero. Con esto se abre otra posibilidad, realizar un backup en un fichero de todos los datos de los elementos de nuestro LVM para futuras incidencias y/o problemas.

5.2 ZFS ("Zettabyte File System")

Introducción

ZFS es un sistema de archivos desarrollado por Sun Microsystems para el sistema operativo Solaris basado en el concepto de grupos de almacenamiento para administrar el espacio físico. Está implementado bajo la licencia Common Development and Distribution License (CDDL). Destaca por las siguientes características:

- Sistemas por su gran capacidad: Los límites de ZFS están diseñados para ser tan grandes que, a la práctica, no se alcanzarán nunca.
- Integración de los conceptos, anteriormente separados, de sistema de ficheros y administrador de volúmenes en un solo producto.
- Innovadora estructura sobre el disco.
- Sistemas de archivos ligeros.
- Administración de espacios de almacenamiento sencilla.
- Comprobación continua de la integridad y reparación automática.

Una de las características que más me ha llamado la atención es una llamada *Dynamis striping*. Consiste en que a medida que se añaden dispositivos al pool, el ancho de las bandas de estos se expande de forma automática para incluirlos, de manera que se utilizan todos los discos del pool para balancear la carga de escrituras entre todos los dispositivos.

Otra característica a destacar de ZFS es que utiliza bloques de tamaño variable hasta 128K. El código disponible actualmente permite al administrador afinar el tamaño máximo de bloque utilizado ya que en ocasiones no es óptimo con bloques grandes. También está contemplado un ajuste automático para adecuarse a las características de la carga de trabajo. Si se activa la compresión se utilizan tamaños de bloque variable, si un bloque se puede comprimir para que quepa en un bloque de tamaño menor, se utiliza el bloque pequeño en el disco, de manera que no sólo se consume menos capacidad sino que se aumenta el volumen de trabajo que fluye a través de la entrada/salida (con el coste de aumentar la sobrecarga de la CPU).

A diferencia de los sistemas de ficheros tradicionales que residen encima de un solo dispositivo subyacente y por lo tanto requieren un gestor de volúmenes separado si se precisa un sistema de archivos mayor que el dispositivo, ZFS se apoya en espacios de almacenamiento virtuales (virtual storage pools). La capacidad de almacenamiento de todos los vdevs está disponible para todos los sistemas de archivos del pool.

Para limitar la cantidad de espacio que un sistema de ficheros puede ocupar, se pueden aplicar cuotas de disco, y garantizar que habrá espacio disponible para determinado sistema de ficheros. Se puede fijar una reserva de disco.

Por último, indicar que ZFS utiliza un sistema de archivos transaccional, es decir, el estado del sistema de archivos siempre es coherente en el disco. Este método hace que el sistema de archivos prevenga daños debidos a una interrupción imprevista de la alimentación o un bloqueo del sistema [20][21].

Definición de términos básicos

Los términos básicos que componen un ZFS son lo siguientes:

- Sistema de archivos: Conjunto de datos de ZFS del tipo filesystem que se monta en el espacio de nombre del sistema estándar y se comporta igual que otros sistemas de archivos.
- Grupo (también llamado pool): Conjunto lógico de dispositivos que describe la disposición y las características físicas del almacenamiento disponible. El espacio para conjuntos de datos se asigna a partir de un grupo.
- Volumen: Conjunto de datos que se utiliza para emular un dispositivo físico. Por ejemplo, puede crear un volumen de ZFS como dispositivo de intercambio.

ZFS se basa en el concepto de grupos de almacenamiento para administrar el almacenamiento físico. El grupo de almacenamiento describe las características físicas del almacenamiento y actúa como almacén de datos arbitrario en el que se pueden crear sistemas de archivos. El tamaño de los sistemas de archivos crece automáticamente en el espacio asignado al grupo de almacenamiento. Al incorporar un nuevo almacenamiento, todos los sistemas de archivos del grupo puede usar de inmediato el espacio adicional sin procesos complementarios.

Comandos

Estos son los comandos básicos para administrar un sistema de archivos basado en ZFS:

- Creación:
  - Grupo: Para crear un grupo de almacenamiento utilizamos en comando *zpool* con el parámetro *create*. La sintaxis es la siguiente:

```
zpool create nombre_grupo disco1 [… discoN]
```

- Filesystem: Para crear un nuevo filesystem utilizaremos el comando zfs del siguiente modo:

```
zfs create nombre_pool/nombre_filesystem
```

El sistema ZFS monta/desmonta automáticamente los sistemas de archivos cuando éstos se crean o cuando el sistema arranca.

- Modificación:
  - Eliminar grupo:
    ```
    zpool destroy nombre_grupo
    ```
  - Añadir disco a un grupo previamente creado:
    ```
    zpool add -n nombre_grup disco1 [… discoN]
    ```
  - Renombrar un grupo:
    ```
    zpool rename nombre_antiguo nombre_nuevo
    ```
  - Eliminar filesystem:
    ```
    zfs destroy filesystem
    ```
  - Modificar/consultar propiedades de un filesystem:
    ```
    zfs get "all" | property[,...]filesystem
    zfs set property=value filesystem
    ```

    Existen numerosas propiedades, pero las más utilizadas suelen ser las siguientes:
    - Para habilitar la compresión de datos dentro del filesystem podemos utilizar la siguiente propiedad: compression=on
    - Podemos determinar el tamaño del bloque de nuestro filesystem con la siguiente propiedad: volblocksize=blocksize
    - Si deseamos limitar el tamaño de nuestro filesystem, podemos establecer un sistema de cuotas con la siguiente propiedad:

      ```
      zfs set quota=size | none nombre_filesystem
      ```

    - O, por ejemplo, proteger un filesystem contra escritura:

      ```
      zfs set readonly=on | off nombre_filesystem
      ```

- Consulta: Para realizar cualquier tipo de consulta sobre nuestro sistema de almacenamiento podemos utilizar el comando *zpool* con los siguiente parámetros dependiendo de la información que deseamos obtener:

  - list: Lista todos los pool de nuestro sistema y su estado. Ejemplo:

```
Artois@opensolaris:~$ zpool list
NAME       SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
bdd_pool  4,97G  3,49G  1,48G    70%  ONLINE  -
rpool     9,94G  3,63G  6,30G    36%  ONLINE  -
```

- status: Muestra detalladamente el estado de nuestro pool. Ejemplo:

```
Artois@opensolaris:~$ zpool iostat
               capacity     operations    bandwidth
pool         used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
bdd_pool    3,49G  1,48G      0      1    329  3,85K
rpool       3,63G  6,30G     23      4  1,36M   101K
----------  -----  -----  -----  -----  -----  -----
```

- iostat: Muestra estadísticas sobre nuestro pool. Ejemplo:

```
Artois@opensolaris:~$ zpool iostat
               capacity     operations    bandwidth
pool         used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
bdd_pool    3,49G  1,48G      0      1    329  3,85K
rpool       3,63G  6,30G     23      4  1,36M   101K
----------  -----  -----  -----  -----  -----  -----
```

- Migración: En algunas ocasiones deseamos guardar toda la configuración de nuestro sistema de almacenamiento, por ejemplo, para realizar una copia del mismo para una máquina gemela a otra podemos realizar exportaciones e importaciones de la configuración de nuestro sistema de almacenamiento mediante el comando *zpool* con los siguientes parámetros: export o import, según el caso.

Cada uno de estos comandos contiene numerosos flags para obtener más información. Todas estas opciones se detallan en la página man de zpool.

Pasos a seguir para implementar un ZFS:

A continuación, se expone un caso práctico de implementación de un ZFS:

1. Análisis y diseño:
   Imaginemos que deseamos crear un servidor de bases de datos con diferentes tipos, por ejemplo, MySQL y Oracle. Para obtener una separación creemos conveniente crear diferentes filesystems. Concretamente crearemos dos llamados mysql y oracle. Cada uno de ellos con una capacidad de 2GB, además, debemos tener en cuanta que en un futuro podemos necesitar más espacio en nuestras bases de datos.

2. Ejecución: Disponemos de dos discos, el c9t1d0 y c9t2d0 de 5GB cada uno. Como con un disco tendríamos de sobras para nuestros filesystems utilizaremos sólo uno de ellos. El otro lo utilizaremos para realizar un mirror para que en caso de error en uno de los discos podamos recuperar el contenido.

   a) El primer paso a realizar es crear el grupo. Para ello utilizaremos el comando *zpool* con el parámetro *create* del siguiente modo:

```
root@opensolaris:~# zpool create bdd_pool mirror c9t1d0 c9t2d0
```

   b) Y, con *zpool list*, podemos observar que se ha creado el nuevo grupo:

```
root@opensolaris:~# zpool list
NAME       SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
bdd_pool  4,97G  74,5K  4,97G     0%  ONLINE  -
rpool     9,94G  3,61G  6,32G    36%  ONLINE  -
```

   Además, con *status* podemos ver el estado en detalle:

```
root@opensolaris:~# zpool status bdd_pool
pool: bdd_pool
state: ONLINE
scrub: none requested
config:

      NAME        STATE     READ WRITE CKSUM
      bdd_pool    ONLINE       0     0     0
        mirror    ONLINE       0     0     0
          c9t1d0  ONLINE       0     0     0
          c9t2d0  ONLINE       0     0     0

errors: No known data errors
```

   Por otro lado, observamos que ya se monta automáticamente:

```
 root@opensolaris:~# df -k /bdd_pool
Filesystem           1K-blocks      Used Available Use% Mounted on
bdd_pool               5128653        19   5128634   1% /bdd_pool
```

c) Llegados a este punto, debemos crear los nuevos filesystems. Para ello utilizaremos el comando *zfs*:

```
root@opensolaris:~# zfs create bdd_pool/mysql
root@opensolaris:~# zfs create bdd_pool/oracle
root@opensolaris:~# df -k /bdd_pool/
root@opensolaris:~# df -k /bdd_pool/*
Filesystem           1K-blocks      Used Available Use% Mounted on
bdd_pool/mysql         5128477        19   5128458   1% /bdd_pool/mysql
bdd_pool/oracle        5128477        19   5128458   1% /bdd_pool/oracle
```

d) Y, por último, aplicaremos un sistema de cuotas para limitar nuestros filesystems:

```
root@opensolaris:~# zfs set quota=2G bdd_pool/mysql
root@opensolaris:~# zfs set quota=2G bdd_pool/oracle
root@opensolaris:~# zfs get quota bdd_pool/oracle
NAME             PROPERTY  VALUE  SOURCE
bdd_pool/oracle  quota     2G     local
root@opensolaris:~# zfs get quota bdd_pool/mysql
NAME            PROPERTY  VALUE  SOURCE
bdd_pool/mysql  quota     2G     local
```

Para verificar que todo es correcto, podríamos hacer dos pruebas. La primera, y totalmente inocua, seria intentar exceder el límite del filesystem haber que sucede. Y la segunda, arriesgada y que nunca debe hacerse si lo datos del filesystem son importantes, poner a prueba el mirror.

Prueba 1:

Para realizar esta prueba lo único que debemos hacer es crear dos ficheros que excedan el límite de nuestro filesystem, por ejemplo:

```
root@opensolaris:~# mkfile 2g /bdd_pool/oracle/mis_tablas.ora
/bdd_pool/oracle/mis_tablas.ora:
initialized 2147090432 of 2147483648 bytes: Disc quota exceeded
```

Sin embargo, si eliminamos la cuota, el resultado es el siguiente:

```
root@opensolaris:~# zfs set quota=none bdd_pool/oracle
root@opensolaris:~# zfs get quota bdd_pool/oracle
NAME             PROPERTY  VALUE  SOURCE
bdd_pool/oracle  quota     none   default
```

Podemos observar que si no establecemos cuota, nuestro filesystem puede utilizar el tamaño completo de nuestro pool:

```
root@opensolaris:~# mkfile 3g /bdd_pool/oracle/mis_tablas.ora
root@opensolaris:~# df -k /bdd_pool/*
Filesystem           1K-blocks      Used Available Use% Mounted on
```

```
bdd_pool/mysql          1982251    512103   1470148  26% /bdd_pool/mysql
bdd_pool/oracle         4616305   3146157   1470148  69% /bdd_pool/oracle
root@opensolaris:~# zpool list
NAME       SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
bdd_pool  4,97G  3,49G  1,48G    70%  ONLINE  -
```

Prueba 2:

Para ejecutar esta prueba hemos desactivado uno de los disco de nuestra máquina virtual y el resultado ha sido el siguiente:

```
root@opensolaris:~# zpool status bdd_pool
  pool: bdd_pool
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist
for
      the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
 scrub: none requested
config:

      NAME        STATE     READ WRITE CKSUM
      bdd_pool    DEGRADED     0     0     0
        mirror    DEGRADED     0     0     0
          c9t1d0  UNAVAIL      0     0     0  cannot open
          c9t2d0  ONLINE       0     0     0

errors: No known data errors
```

Pese a los errores, observamos que todavía tenemos acceso a los datos en uno de nuestros discos. Para añadir un disco a nuestro pool en mirror podemos utilizar el comando “zpool attach bdd_pool disco_viejo disco_nuevo”. Es posible que la replica dure un poco, dependiendo de los datos almacenado y la velocidad del disco.

Conclusión:

Tras analizar el sistema y aprender/adquirir los conceptos básicos he llegado a la conclusión el sistema ZFS es muy flexible. Con pocos conocimientos y pasos se puede crear un sistema suficientemente complejo como para utilizarlo de manera profesional y con la suficiente flexibilidad como para ser modificarlo en el futuro.

6. Diseño

6.1 Pruebas de rendimiento

Llegados a este punto, tras analizar la configuración de ambos sistemas operativos, pasamos a realizar unos tests que emulan, en la medida de lo posible, situaciones reales y habituales que deberán soportar nuestros sistemas informáticos.

En los actuales entornos empresariales encontramos cada vez más aplicaciones basadas en la web. Dichos entornos suelen utilizar servidores web para suministrar dicho servicio. Y la característica más destacable de un servidor web es la capacidad de atender múltiples peticiones simultáneas, normalmente incorporando programas en Java (véase Servlets) basados en tecnologías multithread. Por ello, la primera de las pruebas que realizaremos es el diseño e implementación de un programa multithread sobre el mismo hardware para determinar como gestiona cada sistema operativo el paradigma multithread.

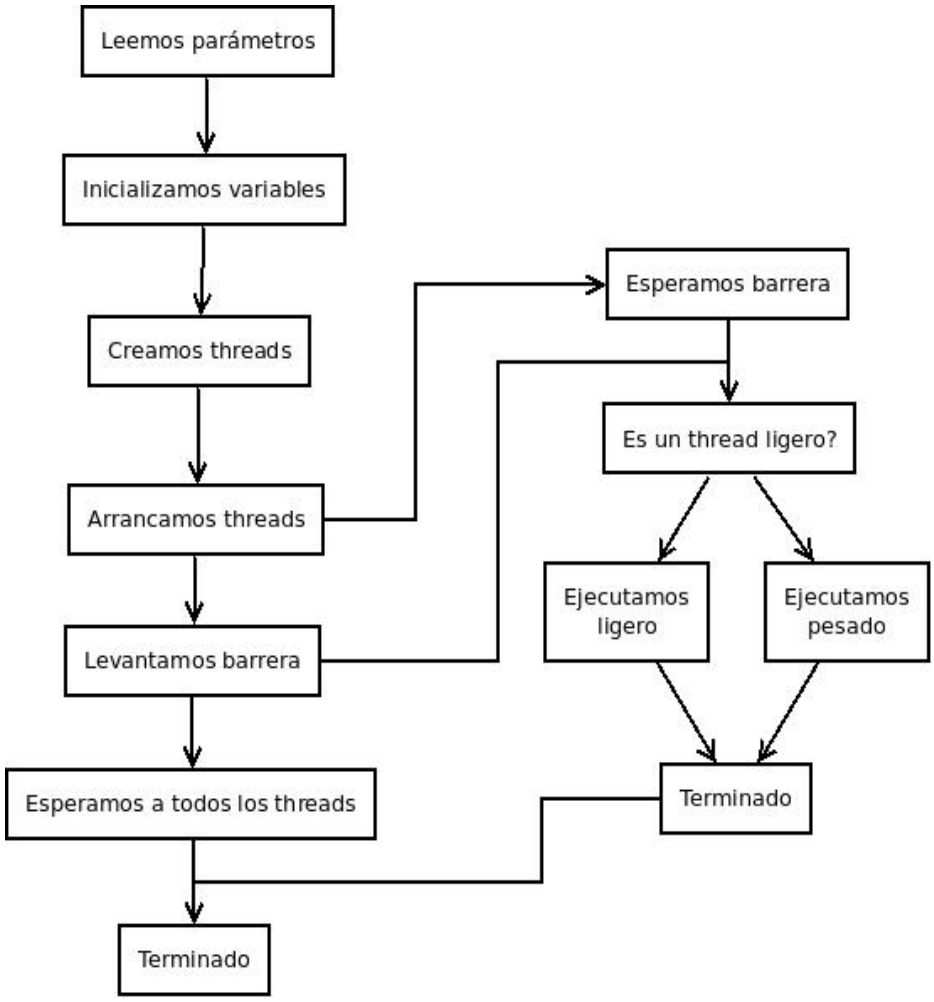
El otro test ha realizar será un clásico en el mundo de la informática, la solución más habitual para almacenamiento de datos estructurados, las bases de datos. En este test realizaremos las acciones más habituales sobre una base de datos inventada y pondremos a prueba la capacidad del sistema operativo para gestionar dichas acciones.

6.2 Multithread

En este test, diseñaremos e implementaremos un programa al cual le transmitiremos por parámetros dos número que indicaran cuantos threads de los siguientes tipos creará:

1. Ligeros: utilizaremos variables del tipo entero para realizar 10^3 operaciones.
2. Pesados: utilizaremos variables del tipo float para realizar 10^6 operaciones.

Con estos dos tipos hemos obtenido dos threads diferenciados con los que podremos realizar diferentes pruebas. La implementación del programa contienen la siguiente estructura:



Antes de pasar a los resultados de las pruebas, me gustaría explicar las tres funciones que he utilizado en la clase creador:

1. La primera a la que llamaremos en el flujo de ejecución será *crear_threads*. Esta función está compuesta por dos bucles que irán creando los threads (ligeros o pesados) según los parámetros indicados. El código es el siguiente:

```
public static void crear_threads(int L, int P, Object o, Thread[] threads){
	for(int i=0;i<L;i++){
		threads[i] = new Hilo(String.valueOf(i), "L", o);
	}
	for(int i=L;i<P+L;i++){
		threads[i] = new Hilo(String.valueOf(i), "P", o);
	}
}
```

2. La siguiente función a destacar es la que arranca todos los threads del vector de threads. El código es el siguiente:

```
public static void arrancar_threads(int L, int P, Thread[] threads){
	for(int i=0;i<P+L;i++){
		threads[i].start();
	}
}
```

3. Por último, y quizá más interesante, es la función esperar. A esta función le pasamos por parámetro el vector de threads para después realizar un bucle donde esperaremos a todos los threads hayan terminado (si es que no lo han hecho ya). La implementación es la siguiente:

```
public static void esperar(Thread[] threads){
		//Esperamos a que terminen los thread
		for(int i=0;i<threads.length-1;i++){
			try {
				if(threads[i].isAlive()){
					threads[i].join();
				}
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Llegados a este punto, vamos a analizar los resultados de las pruebas realizadas.

He divido el test en 5 pruebas y para cada una de ellas he realizado 20 tests para calcular una media. Todas estas pruebas las he automatizado en una shell de Bash para poder ejecutar todos los tests (100) de forma automática y almacenar los datos de forma ordenada.

Las pruebas son las siguientes:

A) Prueba ligeros: Realizamos la ejecución del test con tan solo 10 threads ligeros. En los resultados del mismo podemos observar que la media es similar. Aunque podemos destacar que la diferencia entre el tiempo máximo y el mínimo entre las 20 ejecuciones es menor en OpenSolaris. En la siguiente tabla se muestran los resultados de los test:

	Debian	OpenSolaris
Media	**24**	**23**
Máximo	45	40
Mínimo	18	16
Diferencia	27	24

B) Prueba pesados: Realizamos la ejecución del test con sólo 10 threads pesados. En los resultados que se muestran a continuación podemos ver un incremento de rendimiento aproximado de un 9% en OpenSolaris sobre la ejecución en Debian. Otro detalle a destacar es la menor diferencia entre el máximo y en mínimo en menor en Debian, lo que viene a indicar que el rango de tiempos de ejecución es mucho más reducido que en OpenSolaris, donde existe mucha más diferencia entre los tests realizados. La siguiente tabla ilustra los datos obtenidos en este test:

	Debian	OpenSolaris
Media	**14725**	**13411,5**
Máximo	14844	14658
Mínimo	14599	13311
Diferencia	245	1347

C) Prueba 1000/1000: Realizamos un test mixto donde lanzamos mil threads de cada tipo. Esta es la prueba más equitativa desde el punto de vista de los tipos de thread y la diferencia entre los tiempos de ejecución en ambas máquinas continua en el margen de un incremento de un 9% a favor de OpenSolaris. Los resultados se muestran en la siguiente tabla:

	Debian	OpenSolaris
Media	**1627429**	**1491650**
Máximo	1629591	1588538
Mínimo	1610970	1489006
Diferencia	18621	99532

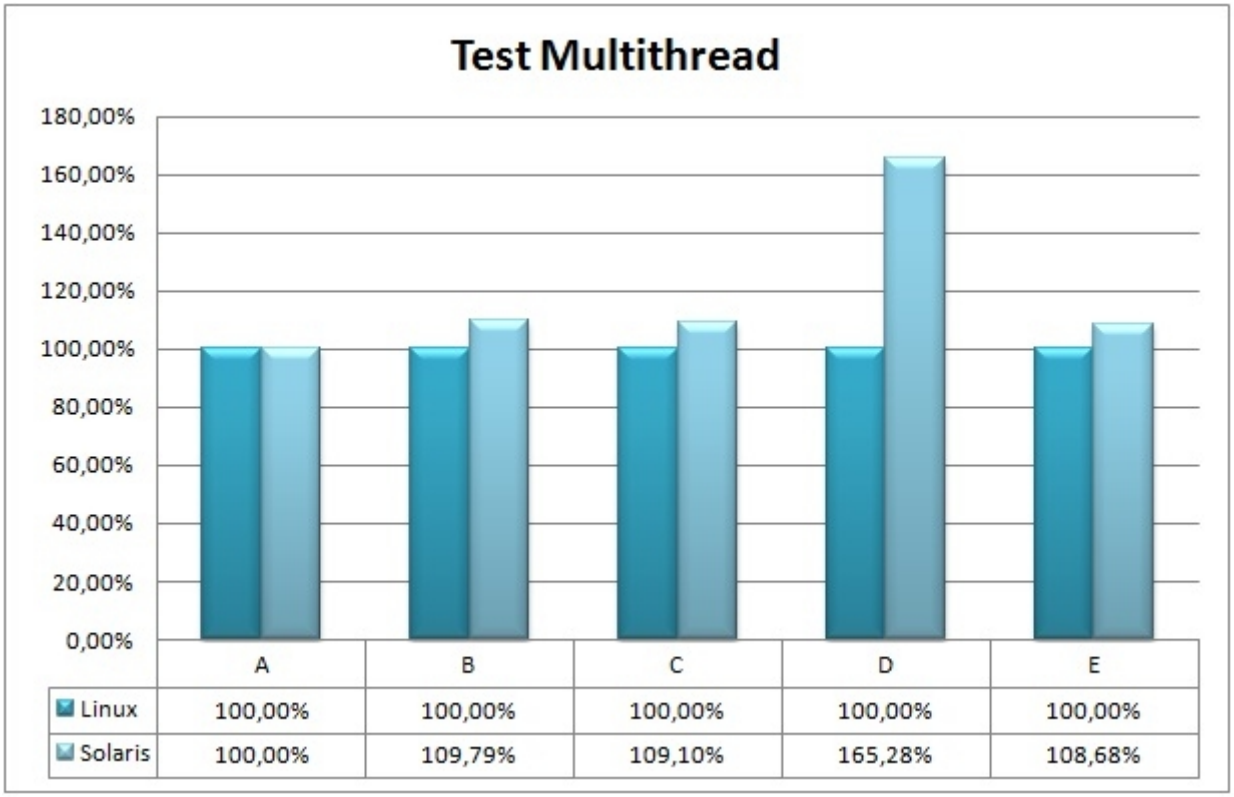
D) Prueba 2000/0: En este test hemos probado la ejecución de thread ligeros y es donde más diferencia hemos notado entre las ejecuciones. Ha resultado que OpenSolaris es un 65% más veloz que el mismo test en Debian. En este caso, y al contrario del resto de pruebas, la diferencia entre el tiempo máximo y mínimo de las 20 ejecuciones es menor en OpenSolaris. En la siguiente tabla vemos los resultados:

	Debian	OpenSolaris
Media	**2349,5**	**1421,5**
Máximo	2856	1548
Mínimo	1901	1255
Diferencia	955	293

E) Prueba 0/2000: En este último test he puesto a prueba el rendimiento de los threads pesados. El resultado obtenido entra dentro de la tendencia de los tests, una mejora en el tiempo de un 8% en el tiempo de ejecución de OpenSolaris respecto a la de Debian. En la siguiente tabla se muestra los datos obtenidos:

	Debian	OpenSolaris
Media	**3250771**	**2991033,5**
Máximo	3256908	3075908
Mínimo	3219071	2978819
Diferencia	37837	97089

A continuación se muestra el gráfico completo de todas las estadísticas de las pruebas:



Haciendo un vistazo rápido al gráfico, podemos extraer la conclusión que en 4 de las 5 pruebas el rendimiento obtenido en OpenSolaris es superior al de Debian. Por otro lado, los resultados más llamativos son los obtenidos en la prueba D. Esto nos puede indicar que alguna de las siguientes (o varias a la vez) pueden ser las causas:

(a) Es posible que OpenSolaris priorice la ejecución de los threads más rápido respecto al resto de procesos, es decir, puede tener un gestor de procesos con una planificación Round-Robin con un tiempo por proceso elevado, lo que implicaría que los threads ligero pudieran acabar con una sola ejecución.

Esta posibilidad podemos descartarla ya que en la ejecución mixta (prueba C) no hemos obtenido un rendimiento tan significativo.

(b) Otra posibilidad es que el kernel de OpenSolaris gestione mejor las operaciones realizadas en los dos tipos de thread con variables enteras y float.

Esta posibilidad podría ser plausible con las operaciones tipo float ya que, tanto en la prueba B como en la E hemos obtenido resultados similares.

(c) Existe otra posibilidad, que la gestión de cambio de contexto cuando un proceso sale de la CPU sea más rápida en OpenSolaris respecto a la de la de Debian.

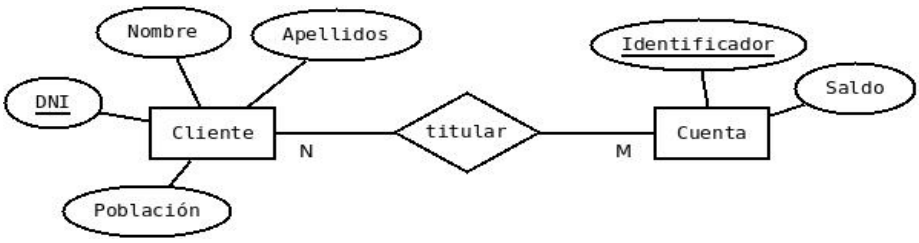
(d) Por último, la última podría ser la más acertada, y es que los servicios por defecto en OpenSolaris consuman menos recursos que los de Debian. A favor de esta hipótesis podemos decir que cuanto mayor es el tiempo que utiliza cada prueba mayor debería ser la diferencia. Aunque mirando mas detenidamente, la prueba D el tiempo de ejecución es inferior respecto al resto (descartando la A).

Cualquiera de las hipótesis anteriores puede ser cierta, incluso una combinación de varias de ellas aunque, en cualquier caso, el claro ganador de este test en OpenSolaris.

6.3 Bases de datos

En esta parte de los brenchmark realizaremos prueba sobre una base de datos. Para ello, he decidido utilizar PostgreSQL ya que, a parte de ser de código abierto, lo hemos utilizado al lo largo de la carrera y estoy muy familiarizado con él. Como entorno para nuestras pruebas he implementado la base de datos correspondiente al siguiente ejemplo.

Imaginemos que tenemos creada una base de datos que corresponde a una entidad financiera, y deseamos realizar unas consultas en una parte concreta de nuestro esquema, los clientes y sus cuentas bancarias. Cada cliente puede ser titular de 0 o más cuentas y una cuenta de tener, como mínimo un titular. Con el siguiente esquema E-R, se muestra la estructura de la base de datos que utilizaremos en nuestros test:



Y, la la definición de las tablas queda del siguiente modo:

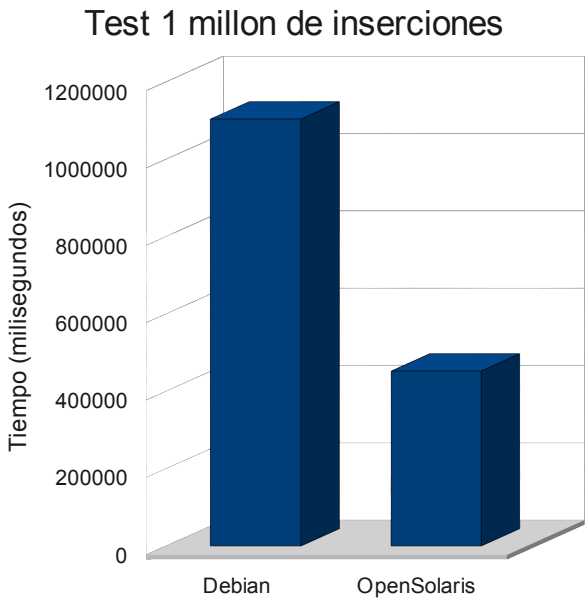
```
CREATE TABLE Cliente(
        dni             integer PRIMARY KEY,
        nombre  varchar(20) NOT NULL,
        apellido1       varchar(30),
        poblacion       varchar(30) NOT NULL
);

CREATE TABLE Cuenta(
        id              integer PRIMARY KEY,
        saldo   integer NOT NULL
);

CREATE TABLE Titular(
        id_cliente      integer REFERENCES Cliente(dni),
        id_cuenta       integer REFERENCES Cuenta(id),
        PRIMARY KEY (id_cliente, id_cuenta)
);
```

La primera de las pruebas corresponde a la inserción de datos, primer paso imprescindible para cualquier base de datos.

Para ello, he implementado un script en Perl para generar un fichero de texto con numerosas inserciones de datos para poder realizar un estudio comparativo en los dos sistemas que estamos utilizando. Concretamente, he realizado dos pruebas: la primera con 1.100.000 inserciones y la otra con 11.000.000. De cada uno se los test, ha sido realizado insertando exactamente los mismos datos, para evitar posibles anomalías, por pequeñas que sean, en nuestras pruebas.

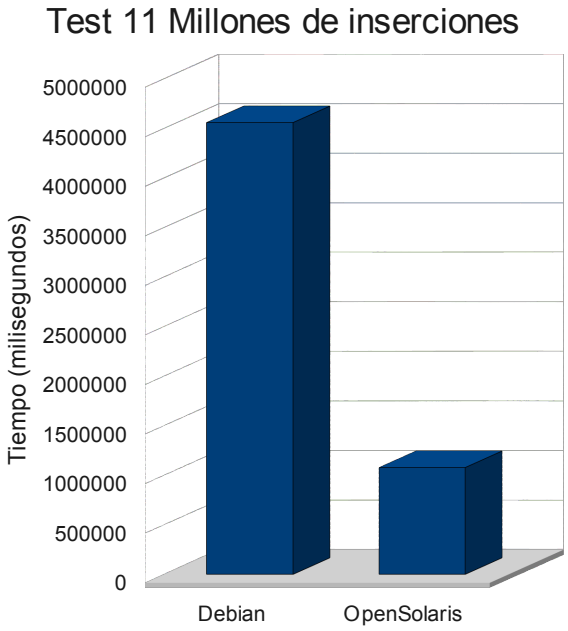
La siguiente tabla ilustra los resultados de las pruebas realizadas, tanto en Debian como en OpenSolaris, en un millón de inserciones:

	Debian	OpenSolaris
Test 1	1104859	452807
Test 2	1103099	451682
Test 3	1103432	452557
Test 4	1104970	452754
Test 5	1104830	452285
Test 6	1103223	451933
Test 7	1107642	452212
Test 8	1105356	451778
Test 9	1104363	456144
Test 10	1103934	453728
Media	**1104596,5**	**452421**



Como podemos observar, el resultado menor se obtienes en OpenSolaris, y es alrededor de un 244% más rápido. Para reforzar los resultado obtenidos, en el siguiente test la diferencia se ve aumentada, al igual que el número de inserciones, si se continua inclinando la balanza a favor de OpenSolaris, pero esta vez con un resultado más abultado:

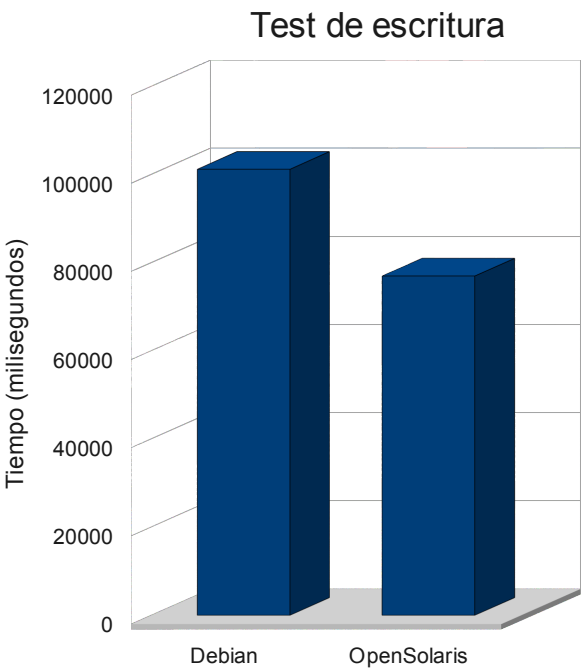
	Debian	OpenSolaris
Test 1	4560025,13	1080017,86
Test 2	4560023,69	1080023,1
Test 3	4560023,98	1080023,43
Test 4	4560025,22	1080024,97
Test 5	4560022,35	1080024,83
Test 6	4560026,03	1080023,22
Test 7	4560024,87	1080027,64
Test 8	4560025,8	1080025,36
Test 9	4560015,22	1080024,36
Test 10	4560021,09	1080023,93
Medias	**4560024,42**	**1080024,15**



En este último caso, resulta que OpenSolaris, respecto a Debian, es 4 veces más rápido. Lo que nos lleva a pensar, junto con el resultado obtenido en la enterior prueba, que la mejor no es ni exponencial ni lineal, pero si se ve aumentada conforme se ven incrementado el número de inserciones. Esto puede deberse a muchos factores, pero normalmente las inserciones ponen a prueba la capacidad de los discos al escribir los datos. Por ello, he realizado un test más para compara los sistemas y pode determinar la los tiempos de escritura en ambos.

Para este test, he realizado un comando que crea un fichero del tamaño indicado. En OpenSolaris es *mkfile*, le indicamos por parámetros el tamaño y nombre del fichero. En Debian es un tanto más complicado, el comando se llama *dd* y los parámetros para crear el fichero son: *dd if=/dev/zero of=prueba bs=1024 count=3145728.*

	Debian	OpenSolaris
Test 1	99712	57435
Test 2	99902	55171
Test 3	101456	60319
Test 4	101746	62824
Test 5	100819	65275
Test 6	106841	64704
Test 7	102842	67918
Test 8	100990	68667
Test 9	101922	70324
Test 10	100203	85106
Test 11	102342	76512
Test 12	102298	77519
Test 13	99960	77631
Test 14	101361	78232
Test 15	102080	82327
Test 16	101686	83417
Test 17	100254	83329
Test 18	100213	84929
Test 19	101014	84046
Test 20	100478	84716
Media	101187,5	77015,5



Los datos obtenidos en ambos test muestran que le rendimiento en escritura es sensiblemente más alto en OpenSolaris, alrededor de un 31% más rápido. Esto confirma la hipótesis de que las inserciones ponen a prueba la capacidad de escritura en disco y que OpenSolaris lo haze con más rapidez.

Por último, he realizado una última prueba sobre nuestra base de datos, una consulta. Las consultas son otra parte vital de nuestras base de datos, ya que permiten extraer de la misma los datos que necesitemos. Los más importante de una consulta es el tiempo que transcurre en ejecutarse, por este motivo, existen numerosos método para optimizarlas y que es ejecuten con más rapidez. Por otro lado, el sistema gestor de base de datos incorporado en PostgreSQL también realiza diversas optimizaciones.

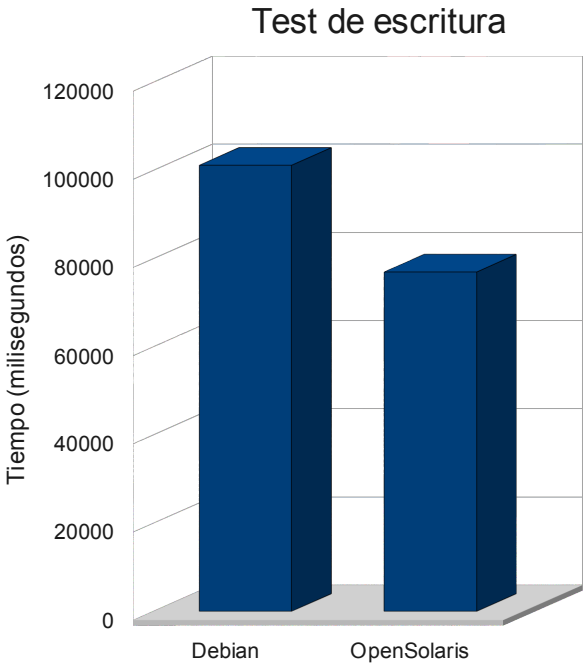
En el ejemplo que tenemos implementado, realizaremos una serie de inserciones (iguales en ambos sistemas) y ejecutaremos una consulta que transcurra a través de todas nuestras tablas, para intentar maximizar el impacto de nuestra prueba. La consulta a realizar es la siguiente:

```
select count(1)
from cliente
where dni in(
      select id_cliente
      from titular
```

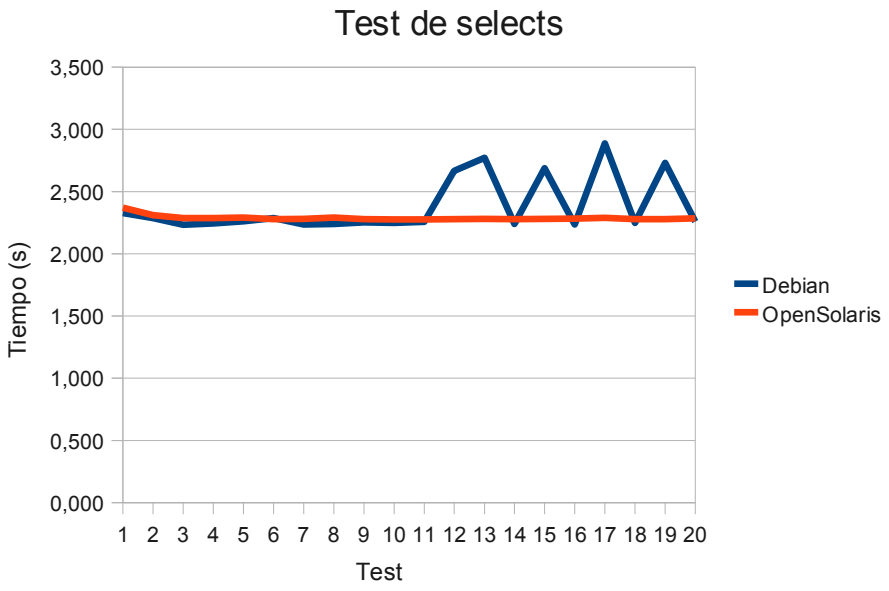
```
where id_cuenta in(
      select id
      from cuenta
      where saldo > 1000));
```

Con los datos que tenemos en la base de datos obtenemos como resultado de esta consulta 297.949 filas. He realizado el test 20 veces para realizar una media más aproximada, con el resultado siguiente:

	Debian	OpenSolaris
Test 1	2,327	2,370
Test 2	2,288	2,310
Test 3	2,234	2,288
Test 4	2,244	2,287
Test 5	2,261	2,292
Test 6	2,287	2,278
Test 7	2,236	2,280
Test 8	2,240	2,291
Test 9	2,252	2,278
Test 10	2,248	2,277
Test 11	2,257	2,277
Test 12	2,666	2,278
Test 13	2,772	2,281
Test 14	2,240	2,278
Test 15	2,688	2,281
Test 16	2,235	2,282
Test 17	2,888	2,290
Test 18	2,249	2,278
Test 19	2,730	2,278
Test 20	2,260	2,285
Media	**2,259**	**2,281**



Como podemos apreciar en la tabla, la media de tiempos es prácticamente un empate técnico, aunque mirando el siguiente gráfico podemos observar unos resultado un tanto elevados en algunas de las pruebas de Debian, aunque la media no se vea muy afectada.



Para terminar el apartado de test me gustaría comentar la gran estabilidad que han demostrado ambos sistemas. Tras diversos abortos en las ejecuciones previas a las pruebas, mientras las ajustaba, los dos sistemas se han recuperado rápidamente, aunque en este aspecto creo que Debian se lleva el primer puesto ya que en más de una ocasión he tenido que reiniciar OpenSolaris para recuperar completamente los recursos mientras que con Debian no.

7. Conclusiones

Creo que una vez concluido el proyecto, lo que todo el mundo espera es que diga que sistema operativo es mejor. Bueno, pues la respuesta es ninguno y todos a la vez. Dependiendo del objetivo final del sistema (servidor de aplicaciones, de bases de datos, etc...) y el tipo de entorno (productivos, investigación y desarrollo, etc...) deberemos elegir nuestro sistema operativo y no existe ninguna prueba puramente empírica que determina cual es el mejor. Hay que tener en cuenta las ventajas e inconvenientes de cada uno y ajustar la decisión dependiendo de las características de nuestro proyecto. Otro punto a tener en cuanta es el nivel de conocimientos / especialización de los administradores de sistemas y personas que los van a utilizar ya que, al fin y al cabo, son los que van a utilizar el sistema.

Una vez concluido el proyecto, podemos decir que sólo hemos tocado la punta del iceberg. Existen numerosas cosa que se pueden investigar mucho más a fondo, p.e. DTrace, una gran herramienta de monitorización que tiene un lenguaje propio. Por otro lado, en este proyecto sólo hemos dado unas leves pinceladas a los sistemas de almacenamiento, tiene algunas otras características y existen muchos otros que no hemos podido incluir en el proyecto, sobre todo, los de licencias propietarias. Por no decir que existen muchos otros sistemas operativos que se utilizan actualmente, tales como AIX, HP-UX y BSD que, perfectamente, podrían haber sido el objetivo de este proyecto.

Por último, indicar que la tecnología más actual parece ser que apunta hacia dos caminos: *Could Computing* y virtualización, y ambos entrelazados. El impacto de estas nuevas tecnologías esta todavía por determinar, pero estoy seguro que abrirá nuevos y espectaculares caminos en cuanto a los sistemas operativos se refiere. Lo que si es seguro es que para nada peligran los sistemas operativos, ya sea para máquinas virtuales o para los sistemas que las albergan, siempre tendremos un sistema operativo.

8. Bibliografía

- [1] http://es.wikipedia.org/wiki/Unix
- [2] http://es.wikipedia.org/wiki/Portable
- [3] http://es.wikipedia.org/wiki/Multitarea
- [4] http://es.wikipedia.org/wiki/Multiusuario
- [5] http://www.top500.org
- [6] http://es.wikipedia.org/wiki/Archivo:Unix_history-simple.svg
- [7] http://es.wikipedia.org/wiki/Historia_de_Linux
- [8] http://es.wikipedia.org/wiki/Núcleo_Linux
- [9] http://es.wikipedia.org/wiki/Debian
- [10] http://es.wikipedia.org/wiki/N%C3%BAcleo_monol%C3%ADtico
- [11] http://es.wikipedia.org/wiki/Jerarquía_de_directorios_en_Linux
- [12] http://es.wikipedia.org/wiki/Proceso_de_arranque_en_Linux
- [13] http://es.wikipedia.org/wiki/Solaris_(sistema_operativo)
- [14] http://es.wikipedia.org/wiki/Opensolaris
- [15] http://docs.sun.com/app/docs/doc/820-2317?l=es&a=load
- [16] http://www.sun.com/software/solaris/availability.jsp
- [17] http://es.wikipedia.org/wiki/DTrace_(Sun_Microsystems)
- [18] http://es.wikipedia.org/wiki/Logical_Volume_Manager
- [19] http://tldp.org/HOWTO/LVM-HOWTO/index.html
- [20] http://es.wikipedia.org/wiki/ZFS_(sistema_de_archivos)
- [21] Guía de administración de Solaris ZFS (http://dlc.sun.com/pdf/820-2314/820-2314.pdf)