Universidad Nacional Autónoma de México

Facultad de Ciencias

SisMo: Un sistema de monitoreo para estaciones de trabajo Linux

T E S I S

QUE PARA OBTENER EL TÍTULO DE
Licenciada en Ciencias de la Computación
P R E S E N T A :

Karla Ramírez Pulido

Director de tesis:
M. en C. José de Jesús Galaviz Casas

2002

A mi mamá,
Ana María Pulido Campos
con todo mi corazón.

Contenido

Lista de Figuras

Lista de Tablas

Agradecimientos

Agradezco primeramente a mi mamá Ana Maria Pulido Campos, a quien he dedicado esta tesis, ya que ha sido la persona que he tomado como ejemplo a seguir en mi vida, por su perseverancia, constancia, entereza, y amor durante todos estos años, ¡me siento muy orgullosa de ser tu hija y te quiero muchísisimo!, gracias mami por todo lo que nos has dado a mis hermanos y a mi. A mis hermanos: Gabriela Paola Ramírez Pulido y Adolfo Antonio Ramírez Pulido que siempre estuvieron apoyándome y animándome desde que los conozco; los quiero muchísimo, y claro sólo quería ser "*amistosa hermanos*". Y ya en serio los adoro. A mi abuelita María González de Pulido que siempre ha estado presente y que sin ella, no se que hubiera sido de mi hoy.

Agradezco de manera muy muy especial a mi novio Francisco L. Solsona Cruz (mi Paquito), por el apoyo, la paciencia, y el amor incondicional que me ha brindado durante todos estos años, (*¡y los que nos faltan!*), y es que eres N [1] detalles. Pero sobre todo, gracias por escuchar en el silencio: `te quiero`.

Agradezco a todos mis maestros desde que empecé a estudiar (hace ya un rato) por compartir su tiempo y algo de sus vidas, en especial quiero agradecer a mis maestros de la licenciatura que me han ayudado a entender y comprender más cosas de las que se pueden imaginar. En especial a mi asesor José de Jesús Galaviz Casas, por su amistad, tiempo, paciencia, y chistes (claro, no podían faltar), ya que sin él esto no hubiera sido posible.

A Elisa Viso Gurovich, una excelente amiga, maestra (la mejor maestra de la carrera en Ciencias de la Computación y sus alrededores), y a la cual admiro muchísisimo. Además de que he tenido el placer de compartir tantas y tan padrísimas pláticas de todo tipo, gracias por ser parte de mi vida.

A Javier García García (Javiercito), uno de mis maestros favoritos, buenísima onda, excelente expositor, sumamente trabajador, y por supuesto un gran amigo.

A Vicky Abrín Batule, padrísima maestra, además de tener la virtud de ser un líder excepcional y de quien tengo el honor de ser su amiga, gracias por el tiempo compartido y por ese carácter tan padrísimo que tienes, en verdad gracias.

A Mónica Leñero Padierna de quien hoy en día tengo el placer de ser su amiga y quien siempre ha creído en mi, además de ser una de las personas más alegres y trabajadoras que he conocido, gracias por ser tan buena onda.

A mis amigas y amigos de siempre y de toda la vida, que han sido lo más alivianado de mi vida, nunca cambien.

A Maricarmen Sánchez Vázquez gracias amiga, por estar siempre en las buenas y en las malas todos estos años. A Ana Judith Moreno Delgado mi amiga la sensishita y carismática, Hugo Vargas Sosa, Patricio Minguela G. por aguantar desde hace ya más de una década, ya que sin ellos no hubiera sido tan divertida la escuela. A Alma Delia Roldán, amiga incondicional de quien sólo puedo tener excelentes recuerdos y hoy en día una padrísisima y sólida amistad.

A mis amiguillos de la licenciatura, por estar todas esas horas conmigo, por esas desveladas de estudio, de relajo, de chismerío y de "viboreo", por todas esas veces que aguantaron, y que sin ellos no hubiera sido lo mismo la carrera, si ... si ustedes son como el

[1] $N \longrightarrow \infty$

sistema operativo en las computadoras, la cerveza en las fiestas, las chispas de chocolate en el helado, la cuerda en el bungee, ... si si son indispensables. gracias a Israel Vázquez Salazar alias el *Flais* por tantas y tantas clases y tareas compartidas, a Iván Hernández Serrano por ser siempre el más divertido y noble de mis amigos, a su hermano Jorge (Simba) por brindarme tan linda y abierta amistad estos años, a Arturo Vázquez Corona mi amigo *Vaz* el más atento y tierno de todos, a Manuel Sugawara Muro uno de mis mejores y más grandes amigos, a Julio Barreiro Guerrero, y Egar A. García C. quiene siempre han estado presentes como grandes amigos. Y claro a mis amigos actuarios Fabiola Alfaro Ramírez, Hugo Díaz Ramírez, Alejandro Sevilla C., Janet Castillo, Vanessa Lavin R. y conexos ya que con ellos las horas de estudio en los cálculos, las álgebras y las geometrías llegaron a ser mucho más ligeritas. También agradezco a mi maestra Margarita Cháves Cano por dejarme tan buena experiencia con la probabilidad y la estadística; a Favio Ezequiel Miranda Perea, por ser un excelente maestro y gran gran amigo, además de consejero incondicional desde ya hace algunos años atrás.

Y `GRACIAS` de nuevo a todos, es decir, a toda mi familia, amigas, amigos, conocidas, conocidos y conexos por brindarme su amistad y cariño.

Capítulo 1

Introducción

El objetivo de este trabajo es elaborar un sistema dirigido a administradores de sistemas, que sea capaz de monitorear el estado de los diferentes equipos de cómputo conectados a una red. El sistema debe ser útil como auxiliar en las labores de administración típicas en sistemas Unix, aunque restringiremos nuestra atención a sistemas Linux en particular. Monitorear significa en este contexto, presentar al administrador de sistemas una visión global del estado de todos los equipos a su cargo, los cuales pueden estar físicamente distantes. Esta información debe poder ser accesible desde un sólo punto y de la manera más gráfica posible. La meta es que el administrador, sin desplazarse de su acostumbrado lugar de trabajo, frente a su computadora, pueda conocer el estado de cada uno de los elementos que componen las diferentes máquinas a su cargo (memoria, sistemas de archivos, usuarios, etcétera), teniendo así los elementos suficientes como para poder tomar las medidas preventivas o correctivas que considere pertinentes y que garanticen el funcionamiento adecuado de los equipos.

El estado global de un sistema está determinado por el estado que guardan sus diversos componentes, en este trabajo en particular los componentes son:

- Memoria (tanto RAM como memoria virtual).
- Procesos registrados en el sistema.
- Carga del procesador.
- Sistemas de archivos.
- Usuarios en sesión.
- Sistemas de archivos compartidos (NFS).
- Conexiones de red.
- Impresoras.

En el resto de este capítulo se plantea con mayor detalle el contexto general del sistema propuesto, los problemas que lo motivaron y que pretende resolver, así como sus cualidades y limitaciones.

1.1 ¿Por qué Linux?

El sistema operativo es el programa más importante que ejecuta una computadora. Se encarga de administrar los recursos de la máquina y permitir el acceso a ellos a las diversas aplicaciones [1]. Contar con un sistema operativo, significa que las aplicaciones pueden ser mucho más compactas, ya que todas comparten el código para acceder al hardware. Toda computadora de propósito general debe tener un sistema operativo que ejecute otros programas. Algunas de las tareas básicas que éste deberá realizar son: reconocer los dispositivos de entrada y salida estándar, mantener directorios y archivos en disco, y controlar diversos dispositivos periféricos, como por ejemplo las impresoras.

No todas las computadoras del mundo tienen el mismo sistema operativo y hoy en día contamos con una gran variedad: Windows95, Windows98, Windows NT, MacOS, Unix, Linux, FreeBSD, entre otros. Esta tesis está enfocada específicamente al sistema operativo Linux.

Linux es un sistema operativo de tiempo compartido, multiusuario y multitarea. Por sus características (ser capaz de una interfaz de usuario amigable, baja demanda de recursos de procesamiento, espacio y versatilidad) se ha convertido en un sistema operativo idóneo para equipos pequeños como las PC's. [2]

Linux se distingue de otros sistemas [7] porque:

- Es multiplataforma, esto es que puede ejecutarse sobre muchas configuraciones distintas de hardware. Mientras que Windows 95 y 98 solo se ejecutan en arquitecturas INTEL [3].

- Es libre, tanto en el sentido económico, ya que no es necesario pagar nada para obtenerlo y usarlo, como en el sentido práctico, es decir que es distribuido junto con el código fuente de Linux y sus aplicaciones. Lo cual es impensable para Microsoft Windows del que solo se distribuyen los archivos binarios, y sólo Microsoft puede actualizar y corregir el código, lo que obviamente limita el grado de adaptabilidad y versatilidad del sistema. Linux es capaz de adecuarse, más o menos fácilmente, a muy diversas necesidades.

- Linux tiene muchas características atractivas y buen desempeño. Esto se debe a que alrededor del mundo muchas personas interesadas en Linux han corregido, actualizado o incorporado nuevas características tanto al kernel [4] como a las aplicaciones. Día con día muchas más personas se incorporan al equipo de desarrolladores, diseñadores y programadores. Todas estas personas encargadas de revisar el código y por supuesto los usuarios, se han dado a la tarea de hacer mucho más eficiente el sistema.

[1] Por aplicaciones entendemos aquí los diversos programas de uso común entre los usuarios: procesadores de texto, hojas de cálculo, juegos, etc.

[2] PC's tradicionales: computadora personal, que no se encuentra conectada en red, usada por una sola persona.

[3] INTEL es una marca registrada.

[4] El kernel o núcleo es la parte más importante de un sistema operativo. Se encarga de la asignación de recursos, de memoria, de decidir qué proceso se ejecuta primero, entre otras cosas, además de controlar y proporcionarle una interfaz a las llamadas del sistema

Linux es estable y confiable, esto significa que el sistema no se caerá de un momento a otro, como usualmente pasa con Windows y no se pierden, tiempo y datos. Su desempeño es mucho mejor que el exhibido por otros sistemas operativos ya que requiere del mínimo hardware para cargarse y ejecutarse. El hecho de ser un sistema operativo libre significa que no se necesita mucho capital para adquirir varias copias. Linux soporta los protocolos de Internet (TCP/IP), además de proveer de herrramientas que facilitan la protección del sistema de usuarios no autorizados. Linux ofrece un ambiente agradable y deseable, particularmente para las personas de la comunidad de programadores, ingenieros en computación o personas afines al área de compilación.

Por estas razones, muchos usuarios de otros sistemas operativos convencionales están cambiando a un sistema operativo mucho más robusto, como Linux, y todos los días muchas personas se incorporan a la comunidad de Linux en todo el mundo.

Linux tiene además muchas más características que le permiten ser considerado hoy por hoy, uno de los mejores sistemas operativos.

1.2 Servicios en Internet y la necesidad de administradores de sistemas.

Internet es una red que conecta a muchas redes de computadoras basándose en el conjunto de protocolos de comunicación, conocidos genéricamente como TCP/IP (*Transmission Control Protocol/Internet Protocol*), que se encargan de regular el intercambio de información entre varias computadoras. Internet ha ido creciendo desde sus inicios. Craig Hunt menciona [6] que el nombre proviene de que fue construida bajo el Protocolo de Internet (*Internet Protocol, IP*). Ahora Internet es un término genérico usado para referirse a toda clase de redes, específicamente las que se conectan a través de los protocolos TCP/IP. Muchas organizaciones alrededor del mundo usan estos protocolos, debido a su gran poderío y seguridad al momento de transmitir cualquier tipo de información:

- Se puede transmitir información independientemente de la arquitectura de la computadora, es decir aunque las computadoras no tengan el mismo hardware y software.
- Se puede integrar a cualquier tipo de red física, con características distintas, es decir es independiente del hardware de la red.
- El manejo de las direcciones en Internet es sencillo.
- Se puede disponer de los distintos programas de aplicación, independientemente de la plataforma de la máquina.

Toda la comunicación en una red es regulada por diversos protocolos. Algunas de las aplicaciones más populares en Internet, basadas en diferentes protocolos son:

- Correo electrónico. Se pueden enviar y recibir mensajes, así como paquetes llenos de información, entre otras cosas, a través de la red, de manera rápida y sencilla.
- Transferencia de archivos. Se pueden copiar archivos de una computadora a otra.

- Acceso remoto. Podemos entrar a otra computadora y trabajar en ella como si nuestra terminal estuviera conectada directamente a ella.
- Grupos de interés (**Usenet**). Se pueden leer y escribir artículos en los miles de grupos de interés de Usenet (también llamados foros).
- Software compartido. Podemos obtener copias de programas de dominio público o de software libre y nosotros mismos podemos compartir nuestros programas con otras personas.
- Acceso a información. Se pueden buscar y recuperar cualquier tipo de información. Si no se está seguro de dónde buscar, existen programas especiales que nos pueden ayudar.
- Comunicación con otras personas. Se puede sostener una conversación con otras personas usando los dispositivos de entrada y salida de diferentes computadoras.

Este tipo de servicios provistos por protocolos de alto nivel, están basados en los servicios más elementales que proveen otros protocolos de nivel inferior. Por ejemplo [3]:

- Servicio confiable de transporte. Se refiere a que los paquetes enviados sean recibidos por la máquina adecuada de tal forma que confime su recepción de mensajes sin anomalías. Al tener un servicio confiable de transporte tenemos también un servicio de transmisión de paquetes sin perder la conexión. Esto se refiere a la entrega completa de los paquetes que han sido transmitidos por una computadora a otra.

Hoy en día los sistemas de cómputo están formados por varias computadoras en red, esto hace referencia a dos o más computadoras conectadas. Las computadoras se conectan en red con el propósito de compartir recursos. Es más rentable tener una impresora compartida por las computadoras personales que tener una impresora por cada máquina. Lo mismo se aplica al software, al disco duro, y en general a muchos de los recursos necesarios en los diferentes equipos. Para que sea posible compartir recursos sin tener un impacto negativo considerable en el desempeño de cada computadora conectada a la red, es necesario garantizar el correcto funcionamiento de la infraestructura física de la red misma y de los equipos conectados a ella. El funcionamiento de un sistema de cómputo requiere no únicamente de computadoras conectadas en red, sino de mantener cada una de estas computadoras funcionando al 100% en cualquier momento, bajo cualquier circunstancia. Cabe mencionar que esto es importante debido a que el sistema de cómputo debe poder recibir y contestar de manera eficiente cada una de las peticiones hechas por las computadoras conectadas en red en el sistema.

El número de computadoras que día a día se van conectando a la Internet se incrementa de manera considerable, de tal forma que el número de redes de computadoras crece y éstas son usadas por millones de personas.

Hoy en día es casi imposible pensar en una computadora aislada, la información y los recursos indispensables para diversas actividades, disponibles a través de una red de computadoras, sólo podrían ser accesibles a un costo muy alto de no poseer la red. Esto ha hecho necesario que en toda red de computadoras exista alguna persona especializada que

realice todas aquellas tareas relacionadas con el mantenimiento y buen funcionamiento de las computadoras, y de la red misma.

El trabajo de estas personas, conocidas como **administradores de sistemas**, es difícil. El trabajo requiere de mucha paciencia, capacidad de análisis y de respuesta, conocimientos del funcionamiento del hardware y del sistema operativo, y sobre todo de experiencia. Cuando ocurre algún tipo de suceso inesperado los administradores deben tomar decisiones rápidamente y que no afecten al sistema y por consiguiente a los usuarios de éste. Los administradores trabajan muchas veces con recursos limitados. Esto, como es de esperarse, hace que no posean todas las herramientas tanto de software como de hardware deseadas. Muchas veces es necesario resolver problemas que nunca antes se habían visto, y que tendrán que resolver, asistidos de algún manual, con la muy conocida técnica de prueba y error, escuchando a personas que ya tuvieron ese problema antes o encontrando la solución con algún golpe de suerte. Hoy en día un administrador de sistemas no se dedica únicamente a instalar el sistema operativo, tiene muchas más actividades como la de planear y diseñar el buen funcionamiento de una red, haciéndola funcional, eficiente, además de mantener todas y cada una de las máquinas o computadoras que estén en el sistema de cómputo a su cargo. Un administrador debe verificar constantemente el funcionamiento, de los equipos, actualizarlos, previendo así cualquier anomalía que éstos puedan tener. Debe además, brindar los servicios que cada equipo requiera para su funcionamiento y esto no sólo es referente a las peticiones de los usuarios, sino también a aquellas partes de software o hardware que el administrador requiera para poder tener su sistema en red trabajando de la mejor forma. El administrador de sistemas tiene a su cargo en términos generales, dos tipos de tareas:

Preventivas: aquellas tareas que realiza el administrador para prevenir ciertas anomalías en el funcionamiento del sistema, por ejemplo si el administrador se da cuenta de que el disco duro de una máquina está casi lleno en su totalidad tendría que tomar ciertas acciones de prevención, como verificar qué es lo que llenó el disco y empezar a borrar algunos archivos que no sean indispensables para el sistema. Es de suma importancia hacerlo para que el usuario que se encuentre ocupando esa máquina, no tenga problemas en el almacenamiento de archivos, así como simplemente que la computadora pueda seguir funcionando.

Correctivas: aquellas tareas que el administrador tiene que realizar una vez que se presentó algún problema. Por ejemplo, si algún usuario rebasara la cuota de disco establecida para su cuenta, el administrador del sistema sería (en principio) el único que puede solucionar el problema, primero quitándole la cuota y decidiendo salvar en algún dispositivo su cuenta o algunos archivos no necesarios para poder volver a poner su cuenta en uso, o bien borrando algunos archivos, y después regresar la cuota al usuario, para que éste pudiera volver a trabajar.

Las redes de computadoras son actualmente imprescindibles. Cada una de estas redes puede estar orientada a distintos propósitos. Actualmente contamos con muchos tipos de redes dependiendo de su arquitectura, tamaño, etc. En este caso prestaremos atención a las redes dependiendo de su tamaño, por lo que se clasifican en dos grandes rubros. Cuando

las computadoras están conectadas directamente (usando algún tipo de cable), el sistema se denomina **Red de Área Local** (*LAN, Local Area Network*) y regularmente se encuentra en un solo edificio. Sin embargo, la conectividad no tiene por qué terminar en una red local. Muchas LAN están conectadas a otras redes, formando lo que se denomina **Red de Área Amplia** (*WAN, Wide Area Network*). Y sólo se tiene cierta diferencia entre una LAN y una WAN dependiendo de la tecnología que se use, por ejemplo:

Siglas	**Nombre**
DAN	*Departamental Area Network* (Red de Área Departamental)
LAN	*Local Area Network* (Red de Área Local)
CAN	*Campus Area Network* (Red de Área de Campus)
MAN	*Metropolitan Area Network* (Red de Área Metropolitana)
WAN	*Wide Area Network* (Red de Área Amplia)

Tabla 1.1: Tipos de redes de computadoras.

Las redes de área local (LAN) se componen de computadoras conectadas a través de dispositivos periféricos, sirven para compartir recursos eficientemente como envío y recepción de archivos, para compartir impresoras, se facilita la comunicación para mandar correo electrónico, etc y fueron diseñadas para operar en un área geográficamente limitada.

Mientras que las redes de área amplia (WAN) son varias redes LAN interconectadas, tienen entonces un área geográfica más amplia, pueden comunicar muchas computadoras a grandes distancias y de esa manera compartir algunos recursos, como archivos e información en general.

El administrador de sistemas Unix es realmente un usuario más, aunque con privilegios de acceso a archivos y programas de configuración que definen el comportamiento del sistema. Gran parte de la labor de un administrador consiste en saber cuáles son y dónde están los archivos y programas que intervienen en la solución de un problema concreto. Esto significa conocer la localización y el formato de cientos de archivos.

La configuración de la red es importante para el administrador del sistema para que éste tenga una idea de las características de la misma. De esta manera, el contar con un administrador de sistemas en red es de suma importancia. Sin embargo no todas las veces podemos asegurar que el administrador conoce al 100% el sistema y todas las peculiaridades que lo conforman. Por esa razón es interesante la posibilidad de contar con algún tipo de herramienta que lo ayude a conocer y garantizar el buen funcionamiento y comportamiento del sistema. El administrador del sistema en Unix como ya dijimos es un usuario más. En Unix el administrador debe conocer cientos de direcciones y archivos, saber dónde se encuentran, qué tipo de información es la que contienen, y cómo se comportan. Además debe ejecutar, revisar y mantener ciertos procesos para asegurar el buen funcionamiento del sistema.

Los administradores deben entonces conocer todas y cada una de las partes que constituyen la estructura del sistema operativo, en este caso un sistema Linux.

La estructura general se encuentra esquematizada en la figura [1]:

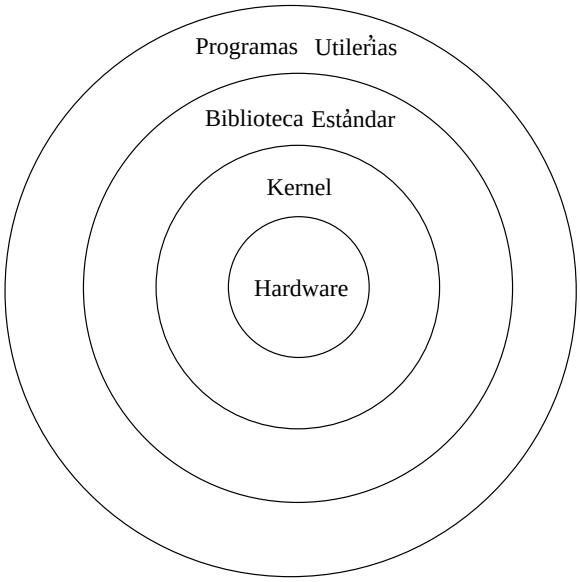

Figura 1.1: Estructura gráfica de un sistema Unix.

Hardware: dentro de este rubro encontramos a las impresoras, el CPU, el monitor, el teclado, el ratón, las bocinas, y los periféricos.

Kernel: también conocido como núcleo, es el encargado de controlar el hardware, administrar la memoria, asignar recursos a todas las actividades que se realizan en la máquina, y de mantener el sistema de archivos.

Biblioteca estándar: éstas son ciertas rutinas escritas (en su mayoría) en lenguaje C. Son las encargadas de crear y manejar los recursos del sistema. Para hacer uso de ellas se hace lo que se conoce como llamadas al sistema.

Programas de utilerías: se refiere a todas aquellas aplicaciones que los usuarios requieren para ciertos propósitos.

1.3 El problema: Mantener un Sistema Linux Vivo

Como se puede ver, administrar un sistema Linux es una tarea ardua y muy compleja. Un administrador capaz de manejar fluídamente varias de las áreas de la computación involucradas en su labor no es fácil de encontrar. Debemos aclarar también que **administrar sistemas** no es una tarea genérica, ya que depende del tipo de sistema de que se trate, para qué se usa, el tipo de usuarios que tiene, el número de éstos, la seguridad requerida por la institución o los usuarios, etc. Es decir, el número de variables a considerar es muy elevado, aún así, nos aventuramos a listar los siguientes como los principales retos a considerar en la administración integral de cualquier sistema basado en Linux:

Usuarios: Los usuarios del sistema son sin lugar a dudas uno de los puntos más delicados en la administración, ya que no sólo debemos considerar la distribución de los recursos y asegurar su óptimo aprovechamiento, sino que además debemos entender las necesidades particulares de cada usuario y tratar de satisfacerlas dentro del marco válido definido por la empresa o institución. Así, es típico monitorear el espacio en disco ocupado por cada usuario, evitar que sus cuotas asignadas sean rebasadas, o por lo menos advertirles que están cerca de su cota superior de espacio.

Debemos revisar que no interfieran con el uso del sistema de otros usuarios, que utilicen programas o aplicaciones que no interfieran o pongan en riesgo la integridad del sistema o de cualquiera de sus usuarios, que su comportamiento no ponga en entredicho la imagen de la institución, etc. En este sentido, cabe aclarar, que los *usuarios* en este contexto se entienden como el comportamiento de los identificadores asignados al personal de la institución y no en sí, a las personas mismas. Esta aclaración tiene sentido y se convierte en un detalle más de la administración, ya que dependiendo de la política de seguridad de la institución, *todo* lo que se "realice" bajo el auspicio de una cierta clave de usuario, es adjudicado a la persona que le fue asignada dicha clave. Pero entonces, si la seguridad del sistema se ve comprometida y una clave de usuario es usurpada y usada maliciosamente por un agente externo, una persona inocente pagará por delitos que no cometió.

Por supuesto, las decisiones particulares, políticas de seguridad y adjudicación de culpa, varía de sistema a sistema y escapan del alcance de este trabajo. Sin embargo, es importante mencionarlas, ya que son un parámetro que, como veremos más adelante, puede ser configurado por el usuario del sistema de monitoreo que nos ocupa en esta tesis.

Procesos: Los procesos no son otra cosa que aplicaciones de tipos y tamaños muy diversos, ejecutados en el sistema bajo el auspicio de una clave de usuario, llamado *dueño* del proceso. Los procesos consumen ciertos recursos, que por ser finitos; por lo tanto, un estricto monitoreo de los recursos y la asignación de los mismos que el kernel de Linux lleva al cabo, son detalles que a pesar de estar (en 90% de los casos) fuera del alcance del administrador del sistema deben ser considerados prioritarios para la administración.

Es el administrador del sistema el encargado de evitar que procesos no autorizados utilicen recursos (en el mejor de los casos, sólo una pequeña sección del tiempo de procesamiento y de la memoria del sistema), o mejor dicho malgasten recursos del sistema. Asimismo, es fácil notar "eventos" raros en el sistema, ya que un monitoreo eficaz y frecuente de los procesos, puede incluso detectar comportamientos anormales en los usuarios del sistema, síntoma casi infalible de que la identidad de dicho usuario ha sido robada y su clave está siendo usada para llevar al cabo tareas no autorizadas, o bien, autorizadas para la persona dueña de la clave, que bien puede no ser la que las está llevando al cabo. En el mejor de los casos, tal robo nunca existió, pero investigarlo también es tarea que podemos considerar *preventiva* del administrador del sistema.

Procesador: El procesador es, como ya mencionamos, asignado para ejecutar distintos procesos por el kernel de Linux; sin embargo, es el recurso fundamental, junto con

la memoria, para ejecutar eficientemente las tareas de los usuarios del sistema (incluyendo las tareas de administración, claro está). En particular, es usual monitorear a intervalos regulares la carga del procesador, para conocer sus horas críticas (horas de mayor afluencia de usuarios y procesos en ejecución), sus horas muertas (horas de baja actividad), etc. Se intenta de esta manera distribuir las tareas que así lo permitan, para alcanzar un mejor aprovechamiento del procesador.

Sistemas de archivos: El sistema de archivos nativo de Linux, conocido como `ext2` a la fecha de este escrito, es un sistema de archivos muy robusto y eficiente. Utiliza un algoritmo para asignación de espacio que reutiliza los "huecos" que se generan al borrar y agregar archivos durante el uso normal del sistema, por lo cual la fragmentación es un problema que no comparte con otros sistemas de archivos de uso común en la actualidad, como FAT32. Sin embargo, a pesar de su buen manejo, el espacio en disco es un recurso finito y el sistema se puede quedar sin espacio para almacenar más información, o bien con espacio, pero sin `inodos` para referenciar más espacio de archivos en el sistema, (para mayor información acerca de los `inodos` refiérase a la sección 5.1).

Es responsabilidad nuevamente del administrador del sistema mantener una estricta vigilancia sobre el espacio, el número de archivos, y demás detalles para asegurar la continuidad en el almacenamiento de la información en el sistema.

Claro que además de particiones en los discos duros del sistema que utilicen `ext2`, pueden existir muchos otros sistemas de archivos conviviendo en el mismo sistema linux, discos fléxibles, CD-ROMs, cintas magnéticas, o incluso otros discos duros, o particiones en ellos, etc. con distintos sistemas de archivos. Si son sistemas usados cotidianamente por los procesos en el sistema, también caen dentro de la responsabilidad del administrador y éste debe ver por ellos, y mantenerlos todos al día.

Memoria: La memoria, es otro dispositivo de almacenaje, igual que los discos duros, o las unidades de cinta que ya mencionamos; sin embargo, en estricta oposición a aquéllos, nos referimos aquí a "memoria" como dispositivos de almacenaje volátiles, es decir, aquellos donde la información no se almacena de manera permanente. En particular, la memoria RAM del sistema es manejada por el kernel de Linux. Como la memoria RAM es limitada y Linux puede atender simultáneamente a muchos procesos, es impensable que todos ellos estén, al mismo tiempo, en RAM. Así que la memoria se divide en dos partes: la memoria física en los chips de memoria RAM del sistema y en una partición (o varias) en los discos duros del sistema, o incluso un archivo en el sistema de archivos de Linux; usualmente llamamos a este espacio *área de swap*. Para los procesos en ejecución, si los datos que requiere están en el swap o en memoria RAM, es un hecho transparente y sin importancia (sin importancia desde el punto de vista teórico, ya que en la práctica si representa una gran diferencia en el desempeño del programa, ya que el disco es varios órdenes de magnitud más lento que la memoria RAM).

El administrador debe también cuidar estrechamente que ningún proceso se apodere de toda la memoria, o de gran parte de ella, ya que eso puede conducir al kernel de

linux a *negar*, o mejor dicho retrasar indefinidamente, la ejecución de otros procesos en el sistema, lo cual es inaceptable en la mayoría de los casos.

Red: Hoy en día es prácticamente impensable un sistema Linux que no esté conectado al menos de manera intermitente, a una red de computadoras, ya sea LAN o WAN (Internet, o redes institucionales de alcance nacional o internacional, por ejemplo). Por tanto, también es importante monitorear el tráfico de paquetes que entran y salen del sistema, poniendo especial atención en el número de clientes que cada servicio en la máquina local puede aceptar y atender de manera adecuada, y también de la relación que éstos guardan con la memoria, el procesador y demás recursos que se mencionaron anteriormente. Nótese que la seguridad, cuando se trata de una máquina insertada en una red, en un ambiente no controlado como por ejemplo en una red universitaria, toma proporciones alarmantes.

NFS: Como sabemos, el sistema de archivos de red nos provee de los mecanismos necesarios para acceder a sistemas de archivos remotos que se encuentran en otras computadoras, como si estuvieran en discos locales. El acceso es transparente para el usuario y los procesos del sistema. Lo que hace escencialmente es montar sistemas de archivos de uno o varios servidores NFS. NFS permite exportar o importar archivos de una máquina servidor a otra. El papel del administrador es sumamente importante, pues depende de él decidir quién puede acceder a este tipo de servicios, si el servidor funciona, si éste se encuentra aceptando las conexiones, si los puertos se encuentran asignados de manera adecuada, si se hizo el montaje de manera correcta, etc., y en caso de cualquier anomalía, corregirla.

Impresoras: Las impresoras son uno de los muchos posibles dispositivos conectados a un sistema Linux, pero al ser una impresora un recurso muy común, se decidió incluirlo como una tarea fundamental en el monotireo del sistema. Administrarla es, nuevamente, mantener un estricto control sobre lo que se imprime y quién lo imprime. Imponer cuotas de impresión, o incluso regular el tipo de documentos, son políticas particulares y pueden variar enormemente de una institución a otra y por eso no las tocaremos aquí. Sin embargo, es importante monitorear el comportamiento de la impresora, el demonio que controla las colas de impresión y detectar errores enviados por la impresora (papel atorado, etc.) y corregirlos oportunamente, etc. Cabe mencionar que las impresoras por tener partes mecánicas, son muy susceptibles de fallo.

1.4 La Solución: SisMo un Sistema de Monitoreo para Servidores Basados en Linux

Como se vió en la sección anterior, los rubros esenciales que deben administrarse en cualquier sistema Linux (no necesariamente son todos, tampoco el orden expuesto es necesariamente el orden de importancia para un sistema particular) son muchos y muy variados. Sin embargo, todos ellos están relacionados, por lo que monitorearlos es una tarea ardua que es dependiente del contexto. En este trabajo, se propone un esquema de monitoreo de todos estos rubros que integra de manera natural un sistema de alerta y prevención de

catástrofes auto-adaptable para sistemas basados en Linux. Sin embargo, el rubro de la seguridad dentro del esquema descrito, en el sentido más estricto de la palabra en este tipo de sistemas, está fuera del alcance de este trabajo.

El sistema de monitoreo SisMo no pretende, por el momento, corregir ningún problema por sí mismo. La intención es proveer al administrador de sistemas de una herramienta de auxilio que le permita ver, desde un solo punto, todo el sistema a su cargo y le avise cuando algo no está evolucionando bien. Sin embargo, sí pretende suplir la inexperiencia o falta de administración que sufren muchos sistemas Linux en la actualidad, apoyando de manera continua y eficiente con información suficiente y señalando inminentes contingencias en un sistema dado. También, por supuesto, los datos que se muestran en el monitor pueden ser utilizados para descubrir exactamente *qué* o *quién* debe ser culpado por la eventualidad. Y dirigir así al administrador a una posible *cura*, aún cuando ésta sea temporal, al problema.

Capítulo 2

Diseño del SisMo

SisMo, nuestro sistema de monitoreo, al igual que la mayoría de los servicios actuales en sistemas Unix en general y Linux en particular, tienen una arquitectura básica cliente-servidor. Esto lo dota de la flexibilidad necesaria para adaptarse a configuraciones de red modernas, ya que el módulo central del SisMo puede estarse ejecutando en una máquina y recibir información de uno (o varios, como veremos más adelante) sistemas remotos, facilitando así la tarea de administración de una red de tamaño pequeño a mediano.

2.1 Modelo Cliente-Servidor

El modelo **Cliente-Servidor** se refiere al tipo de relación que mantienen las partes involucradas (clientes y servidores [1]) en dicho modelo. Los programas que ofrecen uno o varios servicios reciben el nombre de **servidores**, siendo dichos servicios de distintos tipos. Un programa que usa uno o más servicios ofrecidos por un servidor, se llama **cliente**. La comunicación cliente-servidor se lleva a cabo por medio de mensajes.

El modelo tradicional permite organizar aplicaciones distribuidas. Y es a través de procedimientos remotos que se establece la comunicación entre las entidades involucradas en el proceso (computadora cliente y computadora servidor), para poder compartir algunos datos entre ellas. En el modelo tradicional, ciertas computadoras actúan como clientes y solicitan servicios y datos de alguna computadora centralizada conocida como servidor. Los datos solicitados residen en el servidor. Es importante notar que es el programa cliente quien envía una petición al servidor, y espera su respuesta. De la misma forma el programa servidor es el que se encarga de esperar una solicitud y procesarla de tal manera que pueda o no autorizar la petición de servicio.

Este modelo involucra dos partes, la primera donde residen las aplicaciones que serán distribuidas, y en la segunda donde a través de la comunicación entre éstas se solicitan los distintos servicios. De esta manera la comunicación se realiza entre dos computadoras. La Figura 2.1, ilustra el modelo cliente-servidor:

Como se muestra en la figura, el cliente solicita un tipo de servicio al servidor, y éste decide brindarle o no este servicio; para ello entablan cierta comunicación tanto el

[1] En ocasiones el nombre *servidor* se emplea para referirse a una computadora y no a un programa. También se les denomina *host*.

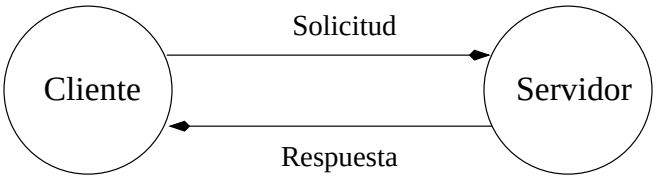

Figura 2.1: Modelo Cliente-Servidor.

cliente como el servidor, regulada por un protocolo. En la figura cada nodo representa una computadora (un cliente y un servidor) y las flechas reflejan la comunicación entre ambas. Es decir la solicitud del servicio (Solicitud) al servidor y la contestación de éste (Respuesta).

Hay distintos tipos de servidores dependiendo del tipo de servicio que proveen, así, es usual tener en una red de tamaño mediano:

Servidor de archivos: provee acceso a los archivos de los usuarios en una red.

Servidor de impresión: controla la cola de impresión de archivos en una o varias impresoras conectadas a la máquina donde el servidor se ejecuta.

Servidor de correo electrónico: mantiene la cola de envío de mensajes: también recibe y distribuye los correos a los buzones de los usuarios del sistema.

La gama de servidores es muy grande; algunos son tan comunes que la mayoría de la gente ha entrado en contacto con ellos alguna vez, como es el caso de los servidores de HTTP y sus clientes (navegadores) corriendo en todo tipo de computadoras, los servidores de HTTP le dan vida a Internet, a tal grado que la gente no versada en computación, incluso llega a pensar que su navegador y los sitios a los que puede acceder *¡son la Internet!*. SisMo también posee procesos de servicio y procesos que usan dichos servicios.

El servidor es el encargado (usualmente) de guardar la información que fluye por los canales de comunicación entre el servidor y sus clientes. Por ejemplo, retomando el ejemplo de la Internet, el servidor es el encargado de guardar la información (en forma de páginas de WEB) que verán los clientes en las ventanas de sus navegadores cuando visiten alguno de los sitios manejados por el servidor.

La comunicación entre clientes y servidores se da por medio de un **protocolo**. Un protocolo está constituido de algunas reglas o normas, que regulan la comunicación entre un servidor y un cliente; en palabras más simples, un protocolo es el vocabulario y la gramática que regula la comunicación. El medio de comunicación es el que utilice la red de cómputo.

Al igual que los servidores, los clientes también pueden ser clasificados, aunque la clasificación es mucho más simple. Por un lado, tenemos los clientes que interactúan con un usuario y los que no. Los clientes que interactúan con un usuario deben ofrecer una interfaz amigable (entendible) al usuario y traducir sus comandos al vocabulario del protocolo para comunicárselos al servidor, recibir la información de regreso del servidor y presentarla al usuario en una forma que pueda comprender. El SisMo tiene varios clientes que no interactúan con el usuario, un cliente especial que muestra los resultados del monitoreo y un solo proceso servidor que sólo provee datos al cliente especial, donde los usuarios en

este caso son administradores de sistemas, y por tanto, la interfaz es altamente técnica, se retomará el tema más adelante.

El funcionamiento general del servidor es:

- Esperar a que algún cliente le haga alguna solicitud. Esto es, usualmente está escuchando de manera permanente, en un puerto de una computadora (a quien conoceremos como el servidor del SisMo). El puerto exacto donde está escuchando es parte del protocolo y tanto servidor como clientes deben conocerlo.

- Cuando un cliente se conecta al puerto donde el servidor del SisMo está escuchando, el servidor inicia una réplica de él mismo (hace un `fork`, (para obtener mayor información acerca de `fork` refiérase a la sección 4.5) [2]). La réplica escucha en un puerto distinto, X. Entonces, el servidor original le indica al cliente que seguirá siendo atendido en X y él se replicará a sí mismo, creando un proceso hijo que difiere del proceso padre sólo en su Identificador de Proceso (PID) y en el identificador del padre (PPID), para seguir esperando comunicaciones de clientes en el puerto original.

- Cuando la comunicación entre la réplica del servidor original y el cliente termina, el canal de comunicación se cierra. El puerto X es liberado y la réplica termina su operación.

De esta forma, la relación que se mantiene es (en teoría) de uno a muchos: un servidor puede atender a muchos clientes simultáneamente. ¿Cuántos? La respuesta depende de muchas cosas, por ejemplo, de la capacidad de la máquina, del tipo de servicios que provee el servidor, los sub-programas que deben ejecutarse, etc. En muchos casos, el número simultáneo máximo de clientes se puede configurar antes de iniciar la operación del servidor.

Un ejemplo de este tipo de relación es **el servicio de impresión**. Por un lado tenemos el servidor de impresión que en este caso es una máquina, la cual proporciona éste servicio a algún cliente que lo requiera. De esta manera el servidor requiere de cierto software que es necesario para poder brindar tal servicio, así como mantener otros tipos de archivos con información para poder operar. Por ejemplo, saber qué usuarios pueden hacer uso de ese servicio, desde qué otras máquinas se puede enviar alguna solicitud de requerimiento del servicio, cómo se irán ejecutando dichas solicitudes, bajo qué condiciones se podrá dar el servicio, etc. Mientras tanto, el cliente requiere de otro tipo de software propio para poder mandar la petición del servicio con el servidor, así como saber si su petición fue aceptada o no, entre otras cosas.

De esta manera el servidor tiene la tarea de dar servicio a todas aquellas solictudes aceptadas (dentro de algún sistema de cómputo conectado a través de alguna red), y para eso necesita contar con algún proceso continuo que se encuentre esperando hasta que haya alguna petición. A este tipo de procesos se conoce como **demonios** (*daemons*). En este caso la aplicación que se encontrará en el servidor será la del servicio de impresión. Entonces el servidor se encarga de estar esperando alguna petición, una vez solicitada éste la ejecuta y le manda una respuesta al cliente.

[2] Función utilizada para crear procesos hijos en el lenguaje de programación C

2.2 Arquitectura del SisMo

SisMo es un sistema cliente-servidor de 3 capas *(tiers)* típico, como se puede ver en la Figura 2.2. En la capa base podemos apreciar las bases de datos donde el SisMo guarda la información que recibe de (la mayoría) de sus clientes y que sirve para calcular las estadísticas que serán mostradas, cuando el cliente especial *admin* las solicite. La capa central la constituye el servidor del SisMo. En esta capa se entrelazan varias actividades, entre otras:

1. El servidor escucha permanentemente por conexiones de los clientes, obj_i, en un puerto determinado, los autentifica y recoge la información que éstos le envían.

2. Controla las bases de datos donde se almacena la información de las máquinas siendo monitoreadas. También tiene la programación necesaria para analizar la información actualizada y transformarla en conocimiento, que será vertido en el cliente del administrador. Dicho conocimiento muestra el estado actual de las máquinas objetivos y sugiere estrategias de solución cuando se presenta algún problema.

La tercera capa está constituida por los clientes. Podríamos pensar que esta capa está dividida en dos, por un lado tenemos a los clientes ejecutándose en las máquinas objetivo que sólo mandan información al servidor y por el otro al cliente que muestra el análisis de la información almacenada al administrador de la red.

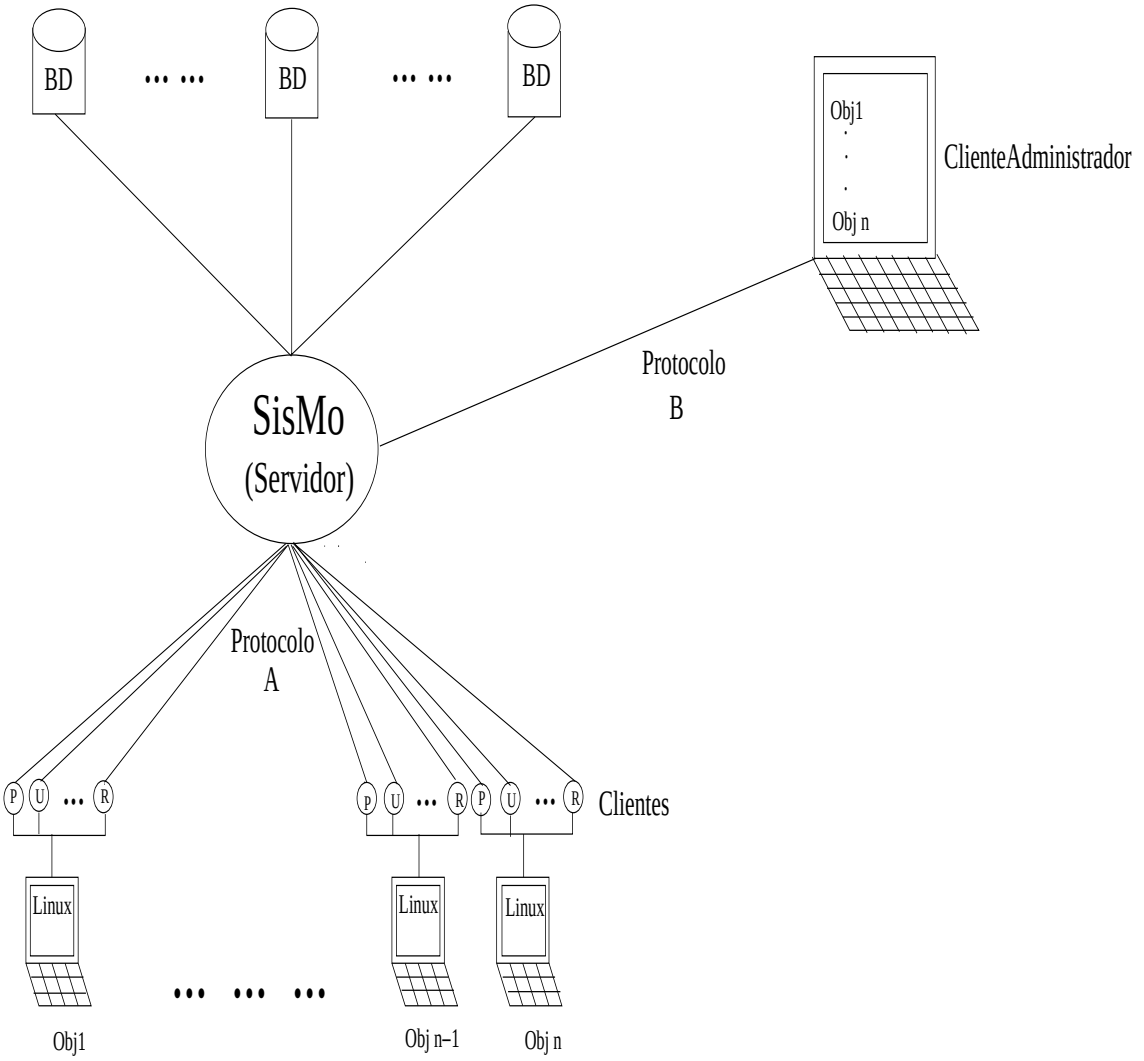

Figura 2.2: Arquitectura del SisMo.

En el caso básico, el SisMo, está diseñado para monitorear un sistema Linux, esto es, el servidor, las bases de datos, los clientes no interactivos y el cliente de monitoreo se ejecutan todos en el mismo sistema. El servidor y los clientes no interactivos idealmente están corriendo de manera automática a intervalos pre-determinados, mientras que el cliente de monitoreo es ejecutado por el administrador del equipo a voluntad.

Como se puede apreciar en la Figura 2.2, el SisMo, puede ser fácilmente extendible para monitorear un gran número de máquinas Linux, digamos todas las máquinas en una red pequeña o mediana, por ejemplo, la red de un instituto de investigación, de un empresa pequeña, etc, donde las máquinas objetivo serían todas las máquinas de los investigadores, o los empleados de la empresa y el cliente de monitoreo es usado por el administrador de red del instituto o la empresa. Ahondaremos en este tema más adelante.

2.2.1 ¿Por qué Linux, por qué Perl?

Mientras que el monitor tiene la tarea de almacenar, analizar y presentar la información sobre el estado de las máquinas objetivo al administrador, de tal manera que éste tenga oportunidad de corregir errores en alguna de las máquinas a su cargo, tomar medidas preventivas, o simplemente verificar que todo va viento en popa, etc., no es claro por qué decidimos centrar el monitor alrededor de máquinas objetivo basadas en Linux. ¿Por qué no cualquier máquina corriendo algún sistema operativo de la familia Unix?

La respuesta tiene dos partes: (1) Porque Linux es libre o *Free Software* y tenemos acceso al código fuente, lo que nos permite analizar cuándo el sistema operativo tiene problemas y definir umbrales de funcionamiento adecuados para el 90% de las situaciones de uso, y (2) porque el uso de Linux es cada vez más usual y competitivo frente al resto de las opciones comerciales de sistema operativo actuales en el mercado. Bien podríamos estar atacando dos o tres sistemas operativos que comparten características con Linux, como OpenBSD, FreeBSD, etc. Sin embargo, muchas decisiones sobre el manejo de memoria, sistema de archivos y asignación de recursos a procesos, son diametralmente opuestas y sería sumamente complicado (y tediosamente innecesario para los efectos de este trabajo) integrar buenos valores por omisión para los umbrales de operación del SisMo. Sin embargo, y esa es una de las razones de usar Perl para programar a los clientes en particular, el código elaborado es 100% transportable con mínimas modificaciones a otras plataformas Unix.

Por otro lado, hemos decido utilizar Perl como el lenguaje de programación para la mayoría de los clientes y el servidor mismo. Entre las ventajas que podemos listar para justificar esta decisión tenemos las siguientes:

1. Perl es un lenguaje de programación muy usado por administradores de sistemas Unix en la actualidad, por lo que el SisMo tiene buenas posibilidades de ser adaptado a medios siempre cambiantes y extendido para acoplarse a un gran número de ambientes de operación fácilmente.

2. Perl es un lenguaje de programación moderno, tiene manejo dinámico de memoria, módulos y un gran número de bibliotecas de soporte.

3. Para manejo de cadenas y en general *streams*, Perl es muy eficiente, tanto que en operaciones simples de entrada y salida puede superar incluso a programas en C. Esto es particularmente útil en el caso de varios de nuestros clientes, porque todo lo que tienen que hacer, es recabar datos de algunos archivos y luego transmitirlos por medio de la red al servidor del SisMo.

4. En prácticamente todos los sistemas Unix podemos encontrar un intérprete de Perl instalado y no sólo eso: Perl fue diseñado para funcionar idealmente en sistemas operativos de la familia Unix y en particular para Linux. Por lo que tenemos a nuestra disposición un intérprete que es difícilmente alcanzado (y prácticamente sin rivales de consideración) en calidad y eficiencia.

Capítulo 3

Manejo de Memoria en el kernel de Linux

El manejo de la memoria para los procesos es de suma importancia. Todos los procesos requieren de memoria, es finita y puede estar bastante limitada, así que el kernel de Linux debe optimizar su uso. Como parte de la estrategia de Linux, en particular, y de Unix en general para administrar la memoria, está el uso de la memoria virtual. Hacer parecer, de modo transparente para los procesos, que la memoria es más grande de lo que es en realidad haciendo uso del disco duro.

En Unix cuando un programa ejecutable es leído en la memoria del sistema por el kernel, se ejecuta y se convierte en un proceso. Tanto en Unix como en todo sistema operativo cada proceso es controlado por el kernel. La memoria del sistema se divide en dos partes [11]: la primera que le corresponde a los **procesos del usuario**, el **espacio de usuario**; la otra parte es la que utiliza el kernel cuando se hacen las llamadas del sistema, es entonces cuando el kernel tiene temporalmente los procesos, conocidos como **procesos del kernel**.

Todos los procesos tienen que estar todo el tiempo en la memoria principal para ejecutarse y el kernel se encarga de mantener las páginas que han sido referenciadas últimamente. La ventaja inmediata es la flexibilidad del mapeo de las páginas en el espacio de las direcciones virtuales, en la memoria física. El manejo a la memoria contempla los siguientes rubros [10]:

Espacios de direcciones grandes: la memoria en muchos sistemas operativos como Linux, tiene la facultad de parecer más grande de lo que es; la memoria virtual es más grande que la memoria física.

Protección: se refiere a la forma en que cada uno de los procesos tiene asignado un espacio en la memoria, el cual es independiente del asignado a los demás, para protegerlo de ser sobreescrito por otros procesos. De esta manera los espacios de dirección virtual son independientes de cada aplicación que se esté ejecutando.

Mapeo de memoria: se refiere a la relación que hay entre las direcciones de memoria y los espacios de direcciones virtuales en la memoria.

Memoria física: se encarga de mantener cada proceso en ejecución.

Memoria virtual: esta es la que mantiene cada proceso ejecutándose en el sistema en un espacio independiente uno del otro, aunque a veces podemos tener procesos que compartan alguna aplicación y es en ese caso cuando hablamos de memoria compartida. Un ejemplo son las bibliotecas que tienen un espacio en tiempo de ejecución, y aún cuando tengamos ejecutándose la misma aplicación varias veces, su código se encuentra en un solo lugar una sola vez y lo que se hace es compartir la misma aplicación.

3.1 Memoria virtual

Cada vez que se ejecuta un programa el procesador crea un proceso, éstos deben de disponer de cierto espacio en memoria. Siempre el procesador va a la memoria por cada instrucción para su ejecución. Cuando un programa es llamado a ejecución por un usuario, su código se carga en la memoria principal (RAM) de donde será leído por el procesador, junto con los datos necesarios para su ejecución. La memoria física de la computadora se divide en páginas. Una *página* es la unidad fundamental que puede ser asignada por el sistema operativo a un programa para almacenar código y datos. Si una página está siendo accedida de manera constante, simplemente se mantiene en la memoria física. Pero puede ser que no sea requerida por un determinado tiempo, en este caso es mapeada a la memoria virtual (en el área de *swap* del disco) donde es almacenada, y una vez que es requerida de nuevo es regresada a la memoria física. El procesador puede regresar cada página a la memoria física dada la dirección virtual que tiene asignada la página en la memoria virtual, es decir una referencia a la dirección de la página. Para hacer esto posible y poder acceder a cada página en la memoria virtual el procesador se encarga de crear una tabla de páginas. La tabla de páginas se compone de dos datos, un *desplazamiento* y el *número de la página virtual (NPV)*, las cuales tienen un número único cada una de ellas.

En la Figura 3.1 se muestra a grandes rasgos el manejo de las tablas para dos procesos que se estén ejecutando, el `Proceso X` y el `Proceso Y`. El primero (el `Proceso X`) cuenta con 10 páginas virtuales, con índices que van del $0 - 9$. En la posición número 0 se encuentra el Número de Página Virtual Cero, denotado por `(NPV0)`. Cada una de estas `páginas virtuales` cuentan con su respectiva `tabla de páginas` también, las cuáles redireccionan el número de página virtual a la memoria física. En el caso del `Proceso X` se puede observar que la `NPV0` es mapeada a su vez en la tabla de páginas y de ahí a la memoria principal en el `número de página tres`, denotado por `NP3`. Y de la misma manera se mapean los `NPV3` y `NPV9` a la `PV0` y a la `PV3` respectivamente.

En tanto, el `Proceso Y` también cuenta con 10 páginas virtuales, con índices que van de $0 - 9$. El `Proceso Y`, en el `NPV1` es mapeado a la `tabla de páginas del Proceso Y` y éste a su vez a la `memoria principal en el número de página cinco`, `PV5`. Al igual que el `NPV7` es mapeado a la `PV2`.

Para poder mapear cada una de las páginas en la memoria virtual, la tabla de páginas es la encargada de decidir en donde ponerlas; de esa manera todas y cada una de las entradas en la tabla de páginas contiene la siguiente información:

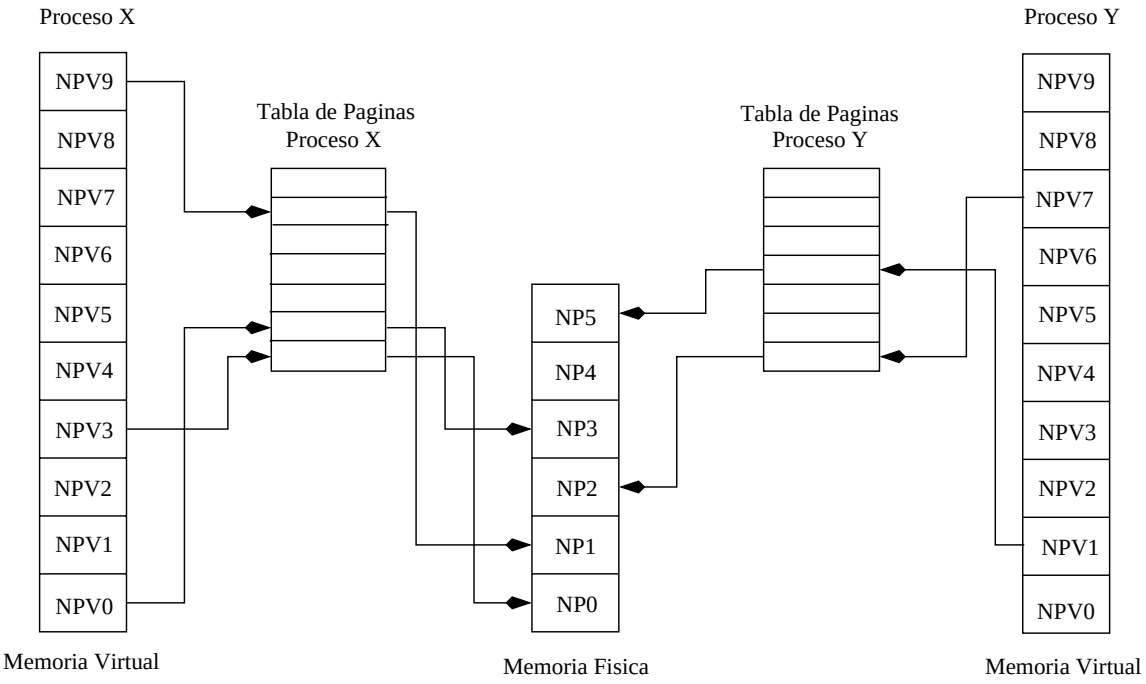

Figura 3.1: Espacio de direcciones virtuales de dos procesos.

- Bandera válida. Se refiere a validar la entrada en la tabla.
- El número de la página física. Que la entrada en la cual se especifica el número de página que que se requiere sea realmente esa página.
- El tipo de página. Se refiere a si la página es de sólo lectura, escritura y/o ejecución.

La tabla de páginas puede mapear cada una de las páginas virtuales en su dirección física usando los números de página como desplazamientos.

3.2 Paginación por demanda

El acceder a un número determinado de páginas con más frecuencia que otras es algo muy común; por ejemplo, al ejecutarse un programa hay ciertas partes de éste que se ejecutan más de una vez y otras que tal vez se ejecuten con menos frecuencia o que nunca lleguen a ejecutarse; es por tal motivo que en el caso del uso de la memoria, si una página ha sido accedida con mucha más frecuencia (que otras), tenga mayor prioridad para seguir en la memoria física y no así en el caso en que no se ha requerido acceder a la página desde hace mucho tiempo. Esta técnica, en la que las páginas más frecuentemente accedidas son mantenidas en memoria física y las otras en la memoria virtual es conocida como **paginación por demanda**.

Cuando el procesador hace referencia a alguna página cuyo número de página no se encuentre en la tabla de páginas, y por consiguiente no se pueda mapear a una dirección en la memoria física, éste mandará un error de "fallo de página".

Otro tipo de falla es cuando algún proceso requiere de alguna dirección virtual que no es suya, y es entonces cuando el procesador manda un error acerca de dirección virtual inválida. En el caso en que el proceso pueda acceder a una dirección de página virtual válida pero esa página no se encuentre en memoria física en ese momento, lo que hace es ir al disco (área de *swap*) a buscar la página requerida, "bajar" de la memoria física alguna otra página que no haya sido usada en el último rango de tiempo al *swap* y poner la página que no se encontraba en su lugar en la memoria física.

3.3 Swapping

Cuando alguna página es requerida por un proceso y el procesador no la encuentra en la memoria virtual ni física, y además no puede ponerla en la memoria física porque no hay el espacio necesario para ponerla (es decir la memoria física esta llena), entonces requiere quitar alguna página de la memoria física y ponerla ahora en disco, y la que es requerida ponerla en la memoria física.

Cuando el contenido de alguna página ha sido cambiado, el sistema operativo tiene que preservar la integridad de los datos ahora modificados, ya que la página puede ser accedida en un futuro. En Linux este tipo de páginas se conocen como **páginas sucias** *(_pages_dirty)*, las cuales son puestas en un archivo especial llamado *archivo de swap*.

Linux usa la técnica de *Least Recently Used (LRU)*. La cual consite en mantener en memoria aquellas páginas que han sido más frecuentemente usadas hasta que se llena la memoria; una vez que se ha llenado y se requiere de otra página que no se encuentra en ésta, el sistema operativo decide cuál página es la que ha estado más tiempo sin ser accedida, es decir la página que ha sido menos usada en el último periodo de tiempo; esto lo hace a través de un contador que tiene cada una de las páginas, el cual se incrementa dependiendo del número de veces que ha sido usada y posteriormente éste la pone en el disco, dejando ese espacio para la nueva página solicitada.

3.4 Control de acceso

Para poder saber qué páginas han sido requeridas recientemente, el sistema operativo consulta la tabla de páginas. En esta tabla se guarda también información acerca del modo de acceso a la página y si son páginas de lectura, escritura o ejecución. En general, para cada página en la tabla se posee la siguiente información:

V *(Valid)*: página válida.

FOE *(Fault On Execute)*: reporta si hubo una falla en alguna ejecución de alguna instrucción, y pasa el control al sistema operativo.

FOW *(Fault On Write)*: al producirse una falla al querer escribir en la página no autorizada.

FOR *(Fault On Read)*: se produce al querer buscar una dirección de alguna página que no se encuentra.

KRE *(Kernel Read Exclusive)*: se refiere a que el código que se está ejecutando sólo puede ser leído en modo kernel.

URE *(User Read Exclusive)*: código ejecutándose en modo usuario es de sólo lectura.

KWE *(Kernel Write Exclusive)*: se refiere a que el código que se encuentre ejecutándose se encuentra en modo kernel de escritura.

UWE *(User Write Exclusive)*: se refiere a que el código que se encuentre ejecutándose, se encuentra en modo usuario de escritura.

***_PAGE_DIRTY*:** se produce si la página necesita ser puesta en el área de *swap* y ha sido modificada desde la última vez que se salvó en el disco duro.

***_PAGE_ACCESSED*:** para saber si la página ha sido usada.

3.5 Cache

Cuando se necesita leer o escribir datos a algún archivo que no se encuentra en memoria, el kernel puede hacerlo [2] pero el tiempo de respuesta del sistema es mucho más lento. Por ese motivo el kernel mantiene en distintos buffers la información que está siendo solicitada. A este tipo de buffers les llamaremos *buffers cache*, Linux cuenta con muchos más tipos de cache en su memoria como:

Buffer cache: este tipo de buffer contiene buffers de datos y todo buffer de datos se encuentra en algún bloque. Los bloques son pedazos de memoria en los cuales se almacenan datos, la información que contienen puede ser leida, o también se puede escribir nueva información en éstos.

Los bloques tienen un tamaño fijo y pueden ser leídos o escritos por algún dispositivo de bloque.

Este buffer es indexado por un identificador de dispositivo y el número de bloque con el fin de encontrar el bloque de datos rápidamente. Los bloques son necesarios para acceder al buffer cache. Si los datos son encontrados en este buffer no es necesario buscarlos en los bloques de dispositivos físicos.

Páginas cache: se refiere a las páginas que se encuentran en la memoria cache, y que a través de su identificador y de un desplazamiento pueden ser encontradas.

Swap cache: aquí se encuentran todas aquellas páginas sucias, que de alguna manera ha cambiado su contenido y son puestas en el área de *swap*.

3.6 La memoria en el SisMo

Todo lo que la memoria puede o no almacenar es controlado por el kernel. En el caso del SisMo es indispensable monitorear la memoria, para saber en qué estado se encuentra, qué tanta carga tiene en un determinado momento, cuánto le queda de espacio disponible para seguir ejecutando procesos, qué tanta memoria compartida se está utilizando y por supuesto conocer el espacio disponible y el no viable del *swap*. La información acerca de la memoria del sistema monitoreado proporciona al servidor parámetros para decidir en qué condiciones se encuentran cada una de las computadoras. Los parámetros que maneja el SisMo) son:

Memoria total: el total de memoria principal (RAM) en el sistema.

Usada: cantidad de memoria usada en ese momento.

Libre: cantidad de memoria libre.

Compartida: cantidad de memoria que está siendo compartida, (bibliotecas de sistema sobre todo).

Cache: cantidad de memoria almacenada temporalmente en la memoria principal pero que no está en uso.

Swap total: cantidad total de memoria *swap*.

Swap usado: cantidad de memoria *swap* que se encuentra usada.

Swap libre: cantidad de memoria *swap* sin usar.

Esta información es una fotografía instantánea de la memoria del sistema al momento que el cliente la envía al servidor del SisMo. Cuando el administrador del sistema lo solicite, el SisMo analizará esta información y sugerirá acciones preventivas, (como resaltar el o los procesos que consumían más memoria) o correctivas (terminar un proceso, cambiar su prioridad, etc.).

Cada una de las computadoras cuentan con un programa cliente hecho en PERL para poder distribuir la información requerida por el monitor.

Capítulo 4

El procesador y los procesos en el kernel de Linux

4.1 El calendarizador y el procesador

La Unidad Central de Proceso (*Central Processing Unit*), a la que llamaremos indistintamente CPU por sus siglas en inglés o procesador, se encarga de las operaciones aritméticas y lógicas y maneja el control de flujo del programa y el flujo de datos, leyendo instrucciones de la memoria y después ejecutándolas.

La operación del procesador está regulada por el reloj del sistema. Éste genera pulsaciones y en cada una de ellas realiza algún tipo de tarea. La velocidad a la que trabaja está dada por la frecuencia del reloj [10]. Un procesador de $100Mhz$, por ejemplo, recibirá $100,000,000$ de pulsos (o ticks) de reloj por segundo[1].

Con base en estos pulsos regulares de reloj, el procesador puede contabilizar el tiempo que un proceso lleva ejecutándose. Como ya se ha mencionado, el procesador debe atender a diversos procesos, divide su tiempo entre ellos asignándole a cada uno una "rebanada" de tiempo, que en Linux es de 50 milisegundos.

Si bien el procesador no es el único elemento del sistema que influye en el desempeño total de éste, sí es el más relevante. Todos los procesos consumen recursos del sistema, el tiempo de CPU, es uno de ellos recursos y por suerte es uno de los más fáciles de manejar[8]:*"Una cantidad constante de procesamiento siempre es viable. En teoría, esta cantidad es* 100% *de ciclos de* CPU*. Un proceso que usa más del* 90% *del* CPU *está en la cota del CPU y está consumiendo más de los recursos viables en el sistema".*

Para evaluar el desempeño del procesador y la carga de trabajo que tiene se deben tomar diversas medidas durante intervalos de tiempo. Por ejemplo:

1. Utilización total.

2. Carga promedio.

[1]Cada tick de reloj, el kernel actualiza la cantidad de tiempo que el proceso actual ha estado en ejecución en el sistema y en modo usuario. Linux también soporta procesos con intervalos de tiempo específicos, conocidos como *llamadas al sistema*

3. Consumo por proceso.

La primera de estas medidas hace referencia la utilización total del procesador en el sistema, lo cual es de gran ayuda para poder identificar si se ha convertido en un cuello de botella o si está subutilizado. La segunda brinda información acerca de la carga promedio del procesador, durante un determinado momento en el sistema. Y el tercero indica los procesos que pudiera estar consumiendo casi todo el tiempo de procesador[2].

También es importante saber qué tipo de tareas son las que mantienen ocupado al procesador y en qué proporción. El tiempo que el procesador está en operación puede clasificarse en tres grandes rubros:

Usuario: se refiere al tiempo que el procesador se ocupa de procesos pertenecientes a los usuarios.

Sistema: se refiere al tiempo que el sistema hace uso del procesador.

No hace nada: se refiere al tiempo en el que nadie hace uso del procesador, y por consiguiente éste está desocupado.

Cada una de las categorías anteriores conocidas por sus siglas en inglés, como *user time (us), system time (sy), idle time (id)*, se encuentran representados por números enteros, los cuales corresponden al porcentaje del tiempo que el procesador ocupa en cada una de las categorías. De tal forma que si el valor del campo del tiempo del usuario es mayor a los valores en los campos, muy probablemente se debe a que la mayor parte del tiempo el procesador se encuentra realizando cálculos, mientras que si los números más grandes aparecen en el campo que corresponde al tiempo del sistema, indican que muchos procesos se encuentran haciendo llamadas al sistema o realizando tareas relacionadas con la Entrada y Salida.

En los sistemas de cómputo más usuales el procesador es único y cada proceso en ejecución debe ser atendido por él. La carga del procesador está necesariemente en función del número de procesos que debe atender. Como ya se ha mencionado, los procesos listos para ejecutarse se encuentran en una cola de atención, donde el proceso al frente de dicha cola es, generalmente, el siguiente en ser atendido. Que la cola de procesos sea larga significa que hay muchos procesos en espera de ser atendidos por el procesador; cuanto más larga sea la cola mayor será la carga del procesador y menor será el desempeño del sistema. Por supuesto que el tamaño de la cola no sólo está en función de la velocidad del procesador, aunque el resto de los recursos utilizados por los procesos también son importantes. Si un proceso hace un acceso a disco y éste es muy lento en su culminación ocasionará que el proceso tenga que aguardar hasta que el acceso sea completado y tendrá que ceder su turno a otros procesos mientras él se vuelve a formar en la cola de espera; si hay varios procesos haciendo lo mismo y además nuevos procesos son creados, la cola tenderá a crecer.

Existen en Linux una gran variedad de recursos disponibles que influyen en la carga del sistema:

- Entrada y salida a disco.

[2]En Linux el comando *vmstat* proporciona información relacionada con la carga del procesador.

- Entrada y salida de la red.
- Tiempo de CPU.
- Procesos marcados (*process slots*).
- Memoria virtual.

Cada uno de estos recursos es administrado por el sistema operativo y cualquiera de ellos puede convertirse en un cuello de botella. Algunos recursos pueden propiciar de manera más rápida esta situación, lo que probablemente se resolverá comprando más o mejor hardware: comprar discos más rapidos, más memoria RAM, etcétera. En general es difícil lograr que el desempeño mejore modificando o reconfigurando el sistema operativo, ya que los núcleos de sistema en general cuentan ya con un desempeño cercano al óptimo.

En síntesis: la carga promedio del procesador se encuentra determinada por los distintos procesos ejecutándose en el sistema, y por lo tanto incluye los procesos esperando por el disco y la entrada y salida de red, no sólo los procesos que se encuentren usando en ese momento el CPU.

En Linux la carga del procesador se puede determinar con el comando *uptime* o también con *ps*; en el primer caso la información que se despliega es el tiempo que ha pasado desde la última vez que se ha inicializado la computadora, el número de usuarios que se encuentren en ese momento, el promedio de trabajos que se encuentran en ese momento en la cola de espera para ser atendidos por el procesador en el último minuto, los últimos cinco y los últimos 15 minutos [14]. Las últimas tres cantidades constituyen una medida *normalizada* de la carga del procesador, por lo que no es directa su interpretación en términos de número de procesos. El máximo valor que pueden alcanzar estos valores, 4, es bastante arbitrario, sin duda, pero es una medida estándar que todos los administradores saben interpretar. La carga del procesador es calculada [3] con muestras de procesos que se encuentran esperando en la cola de procesos a ser ejecutados. Con el comando *ps* se despliega más información acerca de los procesos como:

- Nombre o identificador del usuario dueño del proceso *UID*.
- El identificador del procesos *PID*.
- El identifiicador del padre del proceso *PPID*.
- La terminal desde donde el comando ha sido invocado *tty*.
- El tiempo que el proceso ha estado ejecutándose *REAL CPU TIME* o tiempo de CPU real.

4.2 El procesador en el SisMo

Es de suma importancia verificar que la carga del procesador se encuentre dentro de los límites permisibles, trabajando bajo condiciones "normales", para esto se requiere que

[3] La carga del procesador la calcula el kernel.

el administrador del sistema esté familiarizado con los rangos usuales en cada sistema a su cargo y las posibles consecuencias que acompañan el mal funcionamiento de éste.

SisMo obtiene información en los siguientes rubros. Todos ellos son cantidades medidas durante el periodo de tiempo transcurrido desde el último monitoreo:

Carga: la carga promedio del procesador.

Tiempo de Usuario: hace referencia al tiempo invertido por el procesador en los procesos de los usuarios.

Tiempo del sistema: se refiere al tiempo invertido por el procesador en los procesos del sistema.

Tiempo ocioso: tiempo en el que el procesador ha estado desocupado.

4.3 Manejo de Procesos en el kernel de Linux

Un proceso se define como [10] *"una entidad dinámica que cambia constantemente, conforme sus instrucciones en lenguaje de máquina son ejecutadas por el procesador"*. Para todo propósito práctico un proceso es un programa en ejecución.

Todos los procesos son manejados por el kernel. Este mantiene una tabla de procesos en donde se encuentra información acerca del estado de cada uno de ellos. Cada proceso utiliza distintos recursos del sistema. Si algún proceso termina su ejecución por alguna razón, no necesariamente los demás procesos se ven afectados. En el caso de Linux, por ser un sistema multiproceso, puede ocuparse de distintos procesos "en tiempo real", de tal forma que la Unidad de Procesamiento Central, tiene que maximizar su capacidad de utilización para cada proceso, al mismo tiempo. Así, si algún proceso trata de monopolizar los recursos, el sistema tendrá que decidir si se los brinda o no dado que muchos otros procesos pueden estar en espera de algún recurso que el primero este ocupando. Todo proceso es puesto en una cola de espera en la cual se almacena información relacionada con su estado, su prioridad, etc. Pero, como ya se mencionó, los recursos son limitados y en un sistema capaz de manejar muchos procesos simultáneos, como Linux, resulta curcial asignar los recursos de la mejor manera posible.

En Linux el sistema brinda los recursos solicitados a todos los procesos y puede mantener y darles atención a los que se encuentran en proceso de ejecución al mismo tiempo. Cada proceso se encuentra en una estructura de datos conocida como *task_struct*, donde se almacena información acerca de él. A cada proceso se le puede hacer referencia a través de un arreglo de apuntadores que se encuentran en el vector de tareas *task*. De esta manera el número de procesos queda delimitado por el tamaño del vector de tareas que en el caso de Linux es de 512 entradas. La información contenida en cada una de las estructuras se encuentra relacionada con el estado de los procesos.

Los procesos se pueden categorizar dependiendo del estado en el que se encuentren [10]:

Ejecución: se refiere a todos aquellos procesos que se encuentran ejecutándose en el sistema en un determinado momento.

Espera: se refiere a todos aquellos procesos que se encuetran en estado de espera de ser ejecutados, es decir están en espera de algún recurso.

Parados: se refiere a los procesos que han detenido su ejecución por alguna instrucción del sistema.

Zombies: cuyo control está fuera del alcance del kernel, es decir se encuentran vivos o ejecutándose, pero no son controlados por el sistema.

La parte del sistema operativo encargada de decidir cuáles procesos pueden ser ejecutados y cuáles no es el **calendarizador** o **planificador** (*scheduler*). Cada proceso tiene un **identificador**, que es un número único que lo diferencia de los demás procesos, (estos identificadores no son los índices que contiene el vector de tareas). Además cuenta con identificadores de usuario y de grupo dependiendo del tipo de proceso. De esta forma podemos tener distintos procesos dependiendo del modo en el que se encuentren:

1. **Modo usuario**. Son aquellos procesos que son creados por un usuario del sistema.
2. **Modo kernel**. Son aquellos procesos que son creados por el sistema operativo.

En Linux todo proceso tiene un proceso padre a excepción del proceso inicial de todo el sistema. Todos los proceso en Linux son resultado de duplicar o *clonar* otros procesos. Cada estructura *task_struct* mantiene un apuntador a su proceso padre, a sus procesos hermanos e hijos si es que los tiene. Para poder hacer referencia a cada proceso de manera única, cada proceso cuenta con un *Identificador de Proceso*, (**PID** o **Process Identifier**), que consiste en un número entero asignado por el sistema operativo [4]. A continuación se presenta la información que contiene el vector de tareas:

El **tiempo** desde que ha sido creado, y se va incrementando dependiendo del tiempo que consume de CPU mientras éste proceso viva. Cada vez que un proceso toma posesión del CPU, el planificador le asigna una rebanada de tiempo de 50 milisegundos[5].

El **sistema de archivos**, se encuentra relacionado con los procesos directamente, ya que éstos pueden manipular los archivos como deseen, (por ejemplo, abriéndolos y cerrándolos). Así, todo proceso en Linux tiene información relacionada con el tipo de permisos que tienen los archivos que manipula y con el dueño de los mismos. Algunos de los archivos son de usuarios con ciertos privilegios *(root)* y se espera que ningún otro usuario pueda manipularlos. Para poder de algún modo jerarquizar este tipo de archivos, se les asignan permisos[6].

Los permisos se clasifican en:

- Lectura (R)

[4]En Linux podemos ver los procesos que hasta ese momento tiene el sistema con el comando *pstree*, donde se presenta una estructura de árbol donde el procesos inicial es visto como *init* y sus hijos son todos aquellos procesos que derivan de ése, y si alguno de ellos tiene procesos hijos se deriva otro procesos con sus respectivos nombres

[5]Información obtenida del archivo `sched.c` en el código fuente del kernel 2.4.4 de Linux RedHat 7.1

[6]Son una forma de protección que establece el dueño de los archivos para evitar interferencia en la integridad de éstos

- Escritura (W)
- Ejecución (X)

Cada archivo posee información asociada que permite identificar a su dueño y al grupo al que pertenece (esto se refiere a que algún archivo puede ser accedido no sólo por un único usuario sino por un grupo de éstos que cumplan con ciertos requisitos) y por último posee permisos que limitan el acceso al archivo por parte de su mismo dueño, del resto de los usuarios en su mismo grupo y del resto de los usuarios del sistema. Así que cada proceso que manipule archivos en el sistema deberá conocer los permisos de acceso que éstos tienen.

Un dato que debe mantener el vector de tareas, es acerca de si el proceso es de algún usuario o grupo específico, esta información se mantiene en el campo de **UID** y **GID** (*User Identifier* **Identificador de Usuario** y *Group Identifier* **Identificador de Grupo** respectivamente).

4.4 Planificación de procesos

Como se vió anteriormente cada proceso puede estar en modo usuario o modo kernel. Sin embargo, cuando un proceso hace una llamada al sistema para hacer uso de alguno de los servicios que éste provee, pasa de modo usuario a modo kernel. Muchos procesos hacen llamadas al sistema y es entonces cuándo éste debe decidir qué procesos pueden ocupar determinado tiempo el CPU. Para resolver el problema de asignación de CPU, el sistema le da un tiempo de procesador a cada proceso (la rebanada de tiempo ya mencionada); cuando este tiempo se ha agotado, aunque el proceso no haya terminado de ejecutarse, lo pone en espera y actualiza los campos referentes al estado de cada proceso en el *task_struct*. Así asegura que cada proceso haya tenido el mismo tiempo para ejecutarse y que un sólo proceso no monopolice el tiempo de CPU. Linux utiliza la técnica de **planificación por derecho preferente**, la cual le asigna una pequeña cantidad de tiempo al proceso para ejecutarse, una vez pasado ese tiempo el proceso espera a que se le vuelva a asignar la cantidad de tiempo para ejecución, y otro proceso es ejecutado con la misma cantidad de tiempo; a esta cantidad de tiempo también se le conoce como **porción de tiempo**.

El planificador es el que asigna el tiempo a los procesos ejecutables en el sistema. Cuando se asigna una cantidad de tiempo a un proceso, se hace bajo ciertos criterios de prioridad acerca de los procesos en espera. Todo proceso tiene cierta prioridad sobre otros dependiendo de su estado, el usuario al que pertenezca, etc. La información almacenada para cada proceso que permite la asignación de tiempo con base en prioridades es la siguiente [10]:

Política *(Policy)*: Los procesos se clasifican de acuerdo a cierta política de prioridad y en Linux tenemos dos tipos de procesos:

- Tiempo Real.- Estos tienen la más alta prioridad y son aquellos que tienen que ejecutarse inmediatamente después de que lo han solicitado [7].

[7] Este tipo de procesos tienen un contador diferente al de los procesos normales, para poder distinguirlos. El peso de los procesos normales es el contador y para los de tiempo real es el contador más 1000.

- Normales.- Estos tienen menor prioridad y son básicamente los que pueden esperar cierto tiempo para ser ejecutados.

Las políticas que se aplican dependen del tipo de proceso de que se trate; en el caso de los de tiemop real, se ejecutan bajo la técnica de *Round Robin* (en círculo) o bien *FIFO, First in First Out* (Primero en entrar, primero en salir).

Prioridad *(Priority)*: se refiere a la prioridad que asigna el planificador a cada proceso.

Prioridad de tiempo real *(rt_ priority)*: debido a que Linux soporta procesos en tiempo real, el planificador puede asignarles cierta prioridad relativa.

Contador *(counter)*: es el tiempo que se asigna a cada proceso para ejecutarse. Cada vez que se le da cierta cantidad de tiempo el contador se decrementa en uno.

4.5 El primer proceso

Cuando una máquina se prende, el sistema operativo ejecuta un primer proceso comocido como **proceso inicial** *(init)*. Este comienza por verificar algunos aspectos tanto de hardware como de software, tales como el número de particiones del disco, el tamaño de la memoria, la cantidad de disco usada y la que le queda libre, etc. Para hacerlo, este proceso a su vez crea otros procesos para asignarles distintas tareas dentro del sistema, y esta información se reúne en la estructura del *task_struct* del proceso inicial. Después se ejecuta el programa de inicalización del sistema. El archivo que contiene el programa *init* se encuentra usualmente en el directorio */etc/init*. Éste a su vez usa otro programa */etc/inittab* para inicializar otros procesos.

La idea es separar el primer proceso en varios más que ayuden a las distintas tareas que se tienen que realizar para poder inicializar en primera instancia al sistema, y después estos se clonan para engendrar otros procesos. A este tipo de creación de nuevos procesos a través de una llamada al sistema, que clonan el proceso que la hace, se le conoce como *fork* o clon. Cuando terminan las llamadas al sistema los procesos se ponen en estado de espera para que el planificador los escoja y vuelvan a ser ejecutados. Un nuevo proceso puede ser creado, a partir de alguno que ya existe, el sistema le asigna un único identificador distinto a los anteriores, y mantiene en la estructura de cada proceso el identificador de su padre, debido a que usualmente un proceso hijo no debe sobrevivir a su padre.

La crecación de procesos trae consigo un problema interesante. Cada proceso posee sus propios datos, sus propias variables, en general su propio espacio de direcciones. "¿Qué ocurre con este espacio de direcciones cuando el proceso se duplica mediante *fork*?" "¿Cómo se administra el espacio de direcciones del nuevo proceso en relación con el espacio de direcciones del proceso original?". En esencia se debería copiar el contenido del espacio de direcciones del proceso padre en el espacio del hijo. Para hacer esto de manera óptima se utiliza la técnica de *copy-on-write*. La idea es la siguiente:

Tenemos un proceso original, llamado *X*, que posee su espacio de memoria *mem_X* y sus archivos asociados *files_X*, etc. Este proceso se clona en *X'* (a partir de ese momento existe un nuevo proceso llamado *X'*), el sistema crea las estructuras para la memoria virtual de *X'*, pero en ese momento no "copia" la memoria.

Existen dos casos posibles, a saber: que se requiera leer o escribir en la memoria. Si alguno de los dos procesos, ya sea *X* o *X'* requieren leer de su propio espacio de memoria y ninguno de ellos ha escrito en ella luego del *fork*, entonces de hecho no hay para qué copiarla de *X* a *X'*, porque el contenido es el mismo. Las cosas permanecen así mientras ambos procesos sólo lean de la memoria. Pero si alguno de los dos pretende escribir en su espacio de memoria, entonces sí es necesario copiarla antes de que se lleve a cabo la escritura, para garantizar que cada proceso posea su propia memoria en un estado consistente. Esto se hace porque, a pesar de que *X'* es un clon de *X*, es un proceso diferente y eventualmente hará cosas diferentes a las que hace su padre; sus variables, archivos y datos en general deben estar siempre en un estado acorde con lo que ha hecho él y lo mismo se aplica al proceso que lo engendró.

La ventaja de este esquema consiste en que si nunca, ninguno de los procesos requiere escribir en la memoria, entonces no necesitarán más que una copia y el tiempo y espacio ocupado al hacer la copia se puede aprovechar de mejor manera.

Con frecuencia los procesos deben comunicarse entre sí. Uno de los mecanismos provistos por Unix en general y por Linux en particular para hacerlo es el envío de señales. [13] *"Las señales son análogas en software de las interrupciones del hardware y se pueden generar por diversas causas además de la expiración de los cronómetros"*. A través de las señales pueden enviarse mensajes predefinidos entres dos procesos o del kernel a los procesos. Cada señal tiene un número único que lo identifica y todo proceso tiene un manejador de señales para cada señal. Por ejemplo un proceso suspende el trabajo que está desempeñando y deja de existir, cuando la señal de *kill* es enviada. La reacción de un proceso ante la recepción de una señal específica proveniente de otro proceso o del kernel, está definida por una rutina que el proceso ejecuta cada vez que recibe esa señal. El envío de señales es provisto por el sistema a través de la llamada *kill* que, desafortunadamente, tiene el mismo nombre que la señal de terminación inmediata de proceso. La llamada al sistema *kill* en realidad sirve para enviar cualquier señal y no sólo la que se llama propiamente *kill*.

4.6 Los procesos en el SisMo

La información recabada por SisMo acerca de los procesos en el sistema es la siguiente:

Total: se refiere a la cantidad total de procesos en ejecución en el sistema.

Demonios: se refiere al número de procesos que se encuentran en estado de **demonio** ejecutándose en el sistema.

Sistema: se refiere a la cantidad de procesos en ejecución propios del sistema.

Usuario: el número de procesos en el sistema que pertenecen a usuarios.

Ejecutándose: representan el número de procesos que se encuentran, propiamente dicho, ejecutándose.

Dormidos: se refiere al número de procesos que se encuentran en estado de espera.

Zombies: se refiere al número de procesos que se encuentran en estado *zombie*.

Las cantidades monitoreadas son esenciales para el administrador del sistema, pues le brindan un panorama completo acerca del tipo de procesos que se encuentran ejecutándose en un determinado momento. Con ésta información el administrador puede detectar alguna situación anómala y saber qué tanta memoria y CPU se están consumiendo. Puede además conjugar la información ofrecida respecto a los procesos, con la ofrecida respecto a la memoria en SisMo para tener una idea más clara del estado de los procesos y los recursos que consumen.

Capítulo 5

El Sistema de Archivos en el kernel de Linux

En Linux todos los archivos son mantenidos por una parte del sistema operativo denominada *sistema de archivos*. Éste se encarga de organizarlos jerárquicamente a través de directorios, enlaces flexibles y enlaces duros, entre otros mecanismos, en los distintos dispositivos físicos. Estos últimos se conocen como dispositivos de bloque.

Cuando se inicializan uno o más discos de alguna computadora, se deben definir particiones sobre ellos. Cada una de estas particiones puede tener un sistema de archivos. Existe una gran variedad de sistemas de archivos diferentes, es decir diversas maneras de organizar y localizar archivos en el disco. En el caso de Linux el tipo más conocido y usado actualmente es **EXT2**, el sistema de archivos por omisión. Antes de definir el o los sistemas de archivos que contendrá un disco es recomendable definir una partición de él. Partir un disco duro significa dividirlo en segmentos (normalmente conocidos como particiones aunque el término no es muy afortunado), para clasificar la información que contendrá, cada segmento del disco tendrá luego asociado un sistema de archivos. Es posible, por ejemplo, partir un disco duro definiendo una partición para contener sistemas de archivos del tipo **VFAT**, usado por Windows de Microsoft y otra u otras particiones para contener sistemas de archivos de otros sistemas operativos; EXT2 en el caso de Linux.

El sistema de archivos debe mantener los datos de los archivos, mantener su estructura jerárquica, toda la información de los usuarios y de los procesos del sistema a los que pertenecen los archivos, mantener de manera segura y eficiente esta información, mantener la integridad de los datos y establecer las restricciones de acceso pertinentes. Todo archivo tiene almacenada cierta información [13]:

- Nombre del archivo.
- Tipo de archivo.
- Tamaño del archivo.
- Propietario.
- Información de protección.

- Conteo de uso.
- Tiempo de creación.
- Tiempo de la última modificación.
- Lista de bloques del disco que se usan.

El sistema de archivos de Linux es un módulo independiente del sistema operativo y brinda sus servicios a través del **Sistema de Archivos Virtual**, conocido por sus siglas en inglés como *VFS*, *Virtual File System*. Este permite a los usuarios montar distintos sistemas de archivos. Sin embargo la comunidad de Linux se dió a la tarea de mejorarlo, de tal forma que hoy en día la mayoría de los sistemas se basan en el *Second Extended File System (EXT2)*, el cual es mucho más poderoso y extensible.

5.1 Second Extended File System

El sistema de archivos `EXT2`, [10] guarda la información en bloques de datos, de la longitud fija (1024 bytes). La estructura del sistema de archivos se describe a través de cada uno de sus componentes, es decir los archivos, y éstos se organizan a través de *inodos*. Un inodo mantiene toda la información relacionada con el archivo asociado a él[1]: tipo de archivo, número de bloques que ocupa, permisos de acceso, la hora de creación, de modificación, etcétera.

Cada inodo del `EXT2`, por ejemplo, contiene la siguiente información:

Identificador: el número del inodo.

Tipo: tipo de entrada de directorio asociado al inodo, es decir: archivo regular, subdirectorio, enlace simbólico, etc.

Tamaño: se refiere al tamaño del archivo asociado.

`ctime`**:** sello de tiempo en que fue creado el archivo.

`mtime`**:** sello de tiempo en que fue modificado por última vez.

`mtime`**:** sello de tiempo en que fue accedido por última vez (para leer, ejecutar, etc.)

`dtime`**:** sello de tiempo en que fue borrado el archivo asociado (si es el caso).

Referencias: el número de referencias que tiene el archivo asociado al inodo, es decir el número de "sobrenombres" o enlaces que hacen referencia a él.

Propietario: identificador del usuario al que pertenece el archivo.

[1]Cada archivo tiene asociado un inodo, que es único. La única información acerca del archivo que no es almacenada en el inodo es el nombre; éste sólo aparece en el directorio, donde se indica también el identificador de su inodo.

Grupo: identificador del grupo al que pertenece el archivo. Formalmente hablando es el identificador del grupo al que pertenece el usuario dueño del archivo, pero dado que un usuario puede pertenecer a más de un grupo, debe mantenerse esta información por cada archivo.

Permisos: se refiere al acceso al archivo para el propietario, grupo y el resto de los usuarios.

Bloques de datos *(Data blocks)*: es un arreglo de 12 identificadores de bloque en el disco, aquellos que realmente contienen la información almacenada en el archivo. Si el archivo tiene un tamaño superior a 12 bloques de datos se utilizan bloques de indirección como se describe a continuación.

Como ya se dijo, los primeros doce apuntadores a bloque contenidos en el inodo apuntan a bloques físicos donde se encuentran los datos del archivo asociado a ese inodo. Después se encuentran los últimos tres apuntadores, que tienen hasta tres niveles de indirección de los bloques a los datos, como se muestra en la Figura 5.1:

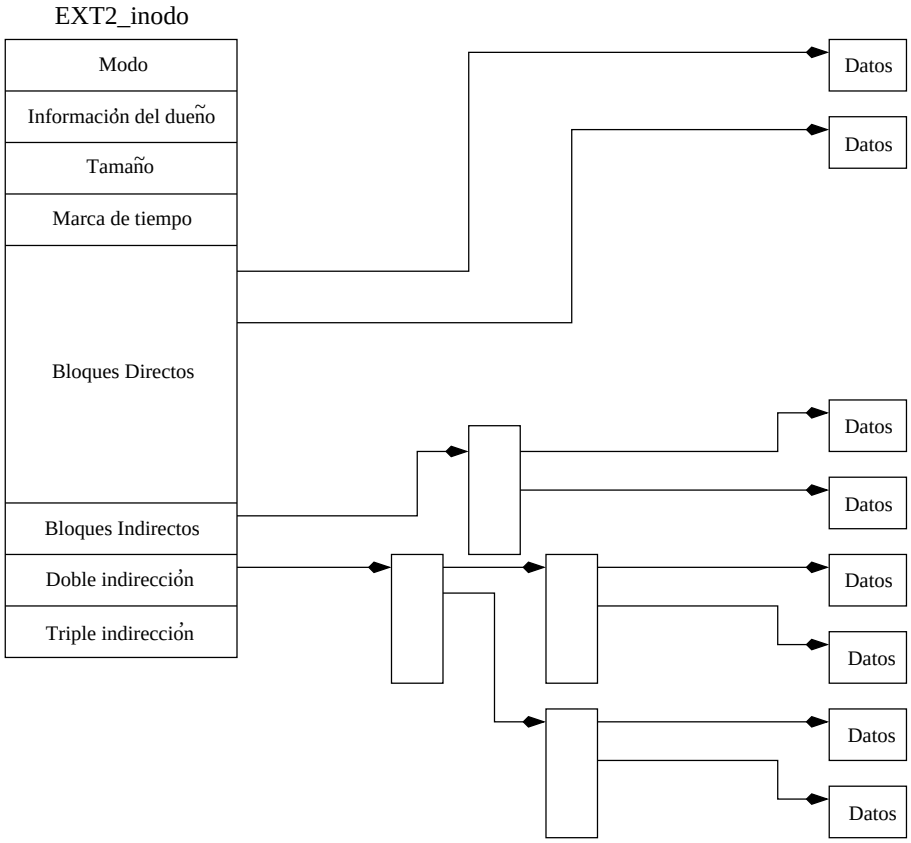

Figura 5.1: Representación de un inodo del EXT2

5.2 El super bloque de EXT2

Cuando se monta el sistema de archivos *EXT2*, se lee el primer bloque del primer grupo de bloques del sistema de archivos. Este bloque se llama el **super bloque**. En él se almacena toda la información que define la estructura general del sistema de archivos, y es tan importante que todos los demás grupos tienen una copia de el, por si se presenta alguna falla que arruine la copia primaria. En el superbloque se almacenan, entre otros, los siguientes datos [10]:

- Número total de bloques y de inodos en el sistema de archivos.
- Número de bloques y de inodos libres.
- Localización del primer bloque de datos.
- Longitud de los bloques (generalmente 1024 bytes).
- Número de bloques y de inodos en cada grupo de bloques.
- Sello de tiempo de la última verificación de este sistema de archivos e intervalo de tiempo entre verificaciones.
- El llamado "número mágico" que identifica la versión del sistema de archivos ($0XEF53$ para un EXT2 normal).
- El estado del sistema de archivos (si se han detectado errores o está "limpio", por ejemplo).
- El número de veces que se ha montado este sistema de archivos desde su última verificación.
- Un indicador de qué hacer en caso de detectar errores en el sistema de archivos.

Cuando un proceso trata de escribir ciertos datos en el disco, el sistema operativo debe permitirle hacerlo en caso de tener espacio suficiente, garantizando que ningún proceso pueda escribir en algún bloque propio de otro proceso. Una vez que un bloque se ha asignado exclusivamente al proceso que lo escribirá, se tiene que verificar que exista espacio suficiente para que los datos puedan ser escritos; si es así, se escriben; en caso contrario se debe escoger algún otro bloque disponible en el sistema donde tengan cabida. Cada grupo de bloques tiene 64 bloques. La búsqueda de un bloque disponible se hace buscando en los bloques contiguos más cercanos, a partir del último bloque que no pudo ser utilizado por carecer de espacio suficiente, en el mismo grupo de bloques donde se escribió por última vez. En caso de que no exista algún bloque disponible en ese grupo, se busca en cualquiera de los demás grupos de bloques, hasta encontrar alguno disponible.

5.3 Sistema de Archivos Virtual (VFS)

El sistema de archivos virtual, conocido por sus siglas en inglés *VFS (Virtual File System)*, tiene a su cargo la tarea de mantener los distintos sistemas de archivos montados para poder accederlos en cualquier instante.

Cuando se arranca el sistema operativo, éste inicializa el sistema de archivos junto con el VFS. Este último mantiene una lista que contiene información acerca de los distintos sistemas de archivos montados en el sistema así como de sus superbloques.

Una cuestión importante es que cuando se compila el kernel, el sistema permite compilar soporte para muchas cosas. En particular cuando se habla del sistemas de archivos, el usuario puede decidir que el kernel soporte muchos sistemas de archivos y no únicamente EXT2. El soporte para diversos sistemas de archivos puede también compilarse en módulos independientes, que pueden añadirse posteriormente al kernel. La posibilidad de incluir soporte para diversos sistemas de archivos y para muchas otras cosas, como módulos independientes que pueden añadirse al kernel con la máquina funcionando, repercute favorablemnte en el desempeño general del sistema. Solo se cargan los módulos cuando se les requiere y no siempre, el kernel es tan pequeño como sea posible y su ejecución es, por tanto, más rápida. Además se tiene un kernel muy flexible, fácilmente adaptable a las circunstancias. La desventaja de este esquema estriba en que si se tiene un kernel muy pequeño y frecuentemente se requiere soporte para varias cosas, el añadir módulos que las soporten hace el sistema menos eficiente que si tuviera por omisión los módulos ya incluidos y añade una carga extra al sistema cuando se pretende descargar un módulo, dado que se debe verificar si no hay proceso alguno que lo utilice.

Es claro entonces que `EXT2` no puede ser un módulo descargable, porque siempre se debe dar soporte a ese sistema de archivos.

Para acceder a los distintos archivos y directorios del sistema, las rutinas del sistema recorren los inodos del VFS, hasta encontrar el archivo que se estaba buscando, en caso de que exista. Los inodos que han sido encontrados son puestos en la *antememoria* o *cache*; de esa forma todos los inodos de los archivos que recientemente han sido usados se dejan en el cache, para que la siguiente vez que sean requeridos, el acceso a ellos sea mucho más rápido. El VFS también mantiene un cache para los directorios, en donde, al igual que para los archivos, se mantienen los inodos de los directorios que hayan sido solicitados con más frecuencia recientemente.

El sistema de archivos de Linux se conforma comúnmente como un árbol de directorios como se muestra en la Figura 5.2:

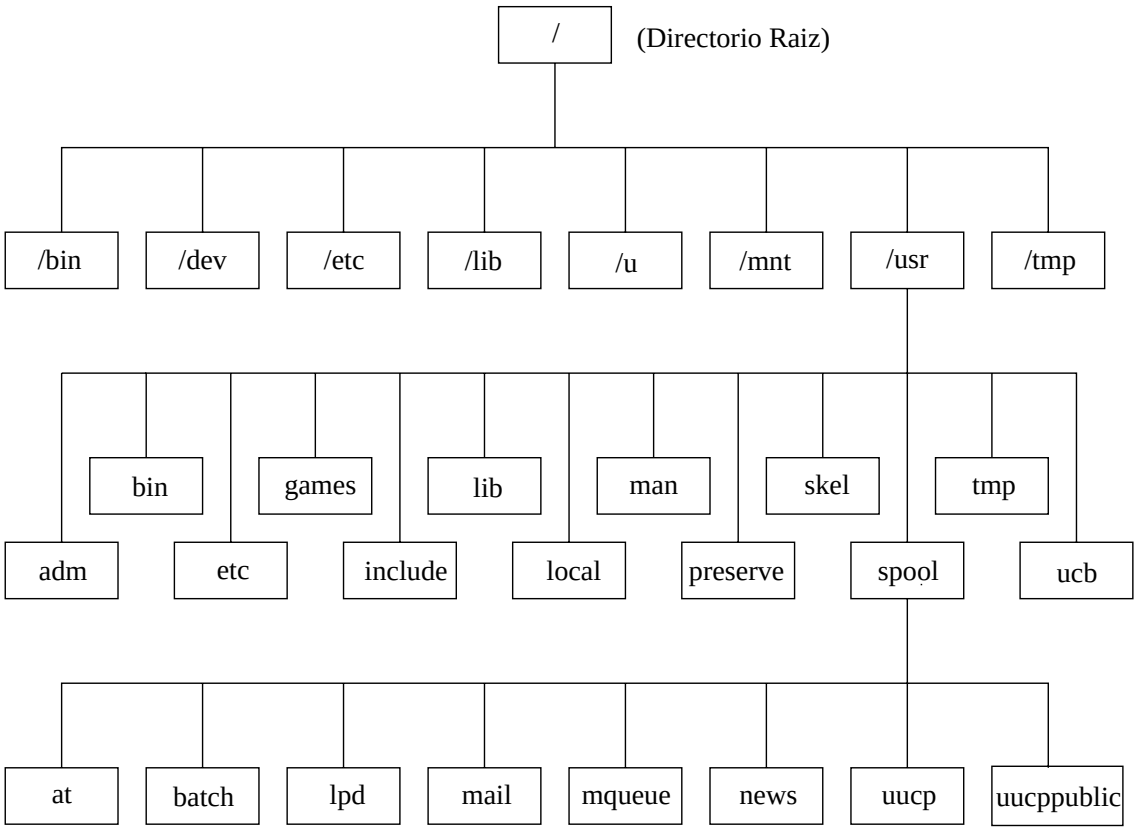

Figura 5.2: Representación del sistema de archivos EXT2

5.4 SisMo en el sistema de archivos

Los sistemas de archivos son, pues, una pieza fundamental del sistema en su conjunto y por ello se debe monitorear su estado con frecuencia. El sistema de monitoreo SisMo registra la siguiente información respecto al sistema de archivos:

Sistema de archivos: cuántos y cuáles sistemas de archivos se encuentran montados en el sistema.

Bloques: cantidad de bloques asignados a un sistema de archivos.

Usado: cantidad en espacio utilizado en un sistema de archivos.

Viable: cantidad en espacio disponible en el sistema de archivos.

Porcentaje: porcentaje de espacio utilizado por cada uno de los sistemas de archivos, montados en el sistema.

Punto de montaje: se refiere al directorio donde se ha montado el sistema de archivos.

Esta información proporciona una buena visión general del sistema de archivos en cada máquina. El administrador puede entonces tener información acerca de si el sistema de archivos esta montado en el directorio adecuado, además de conocer el tamaño de los distintos sistemas de archivos y saber cuánto espacio en disco están utilizando, y por supuesto cuánto espacio queda disponible. Esto es de suma importancia dentro de cualquier sistema. Si en un determinado momento el espacio en disco de alguna máquina estuviera usado en un 99% de su capacidad total, sería indispensable que el administrador verificara qué tipo de procesos están consumiendo su capacidad y corrigiera esta situación.

Capítulo 6

Los usuarios en el kernel de Linux

En Unix en general y en Linux en particular cada usuario tiene asociado un número único que lo identifica en el sistema, denominado **identificador de usuario** *(User Identifier* o *UID)*. Para facilitar la identificación de los usuarios por parte de otros usuarios, cada `UID` se asocia a una cadena de carácteres conocida como el `login` del usuario. Así por ejemplo la persona llamada *Adolfo Pulido* puede tener asociado el `UID` 750 que es equivalente al login *apulido*. El sistema operativo registra los archivos creados por *Adolfo* como el usuario 750 y cuando otro usuario pregunta quién está en sesión, si está *Adolfo*, se le dice que está *apulido*. Todo usuario debe pertenecer por lo menos a un grupo en el sistema [1]. Los grupos en Linux se encuentran registrados en el directorio */etc/group* y se ven como sigue:

```
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
```

El primer campo nos dice el nombre del grupo, separada por dos puntos; sigue la contraseña del grupo cifrada, seguidos del número (identificador) del grupo; y por último, separados por comas, los usuarios integrantes del grupo.

El identificador del grupo (GID **Group Identifier)**, es también un número entero. A continuación aparecen algunos de los grupos que usualmente se encuentran en un sistema:

GID	**Nombre**	**GID**	**Nombre**	**GID**	**Nombre**
0:	root	1:	bin	2:	daemon
3:	sys	4:	adm	5:	tty
6:	disk	7:	lp	8:	mem
9:	kmem	12:	mail	13:	news
14:	uucp	15:	man	19:	floopy
20:	games	50:	ftp	100:	users

Tabla 6.1: Algunos de los identificadores de grupo más conocidos.

[1] Un usuario puede a lo más pertencer a 16 grupos.

El sistema mantiene una lista de los usuarios en el archivo */etc/passwd* donde se encuentra la información relacionada con los usuarios, entre otras cosas: su login, su UID, el identificador de grupo al que pretenece, la ubicación exacta de su directorio personal de trabajo (home) e información adicional opcional como su nombre real.

Cada vez que se crea un nuevo usuario, el sistema y el administrador deben efectuar ciertas tareas como [14]:

- Editar el archivo */etc/passwd*.
- Editar el archivo */etc/group*, en caso de que el grupo asociado al usuario que se esté dando de alta no exista, y deba agregarse. De existiir debe añadirse al nuevo usuario como integrante de el o los grupos a los que pertenece.
- Crear un directorio personal para cada usuario: el *home* del usuario.
- Copiar los archivos de inicialización en el directorio personal.
- Asignarle una contraseña al usuario, para que pueda ingresar de manera segura a su cuenta dentro del sistema.

Actualmente Linux cuenta con un comando para hacer todo esto automáticamente, *addusser*, el cual crea el directorio del usuario de manera automática y actualiza el archivo */etc/passwd*. Los archivos de inicialización o configuración se utilizan para que el usuario tenga lo mínimo indispensable para poder empezar a trabajar en el ambiente. Algunos de los archivos que se copian en el directorio del usuario son:

Archivos de configuración del intérprete de comandos: Se refiere al intérprete que utilizará el usuario en el ambiente. Por default se propone *Bash*, aunque Linux cuenta con muchos más como *Bourne shell, C shell, TC shell, Korn shell, Z shell*, entre otros. Todos tienen sus propios archivos de inicialización de *login*, y los que ejecutan un nuevo intérprete de comandos cada vez que se abre uno de éstos. A continuación se muestra una pequeña lista de los intérpretes más utilizados:

Nombre	Shell	Script Login	Script nuevo shell
Bourne Again Shell	bash	.profile	.bashrc
Bourne shell	sh	.profile	ninguno
C shell	csh	.login	.cshrc
TC shell	tcsh	.login	.tcshrc
Korn shell	ksh	.profile	.kshrc
Z shell	zsh	.zlogin	.zshrc

Tabla 6.2: Diferentes intérpretes de comandos soportados en Linux.

Archivo .exrc: para poder usar el editor *vi*.

Archivo .emacs: aquí se reune la información suficiente para poder utilizar el editor *emacs*.

Archivo .newsrc: especifica los grupos de interés nuevos.

Archivo .Xdefaults: para poder configurar el ambiente de ventanas.

Archivo .xnitrc: se especifica el ambiente gráfico inicial.

Para dar de baja a un usuario en el sistema el administrador debe, de manera general, realizar las siguientes tareas:

- Quitar la entrada del usuario del archivo */etc/passwd*.
- Quitar el nombre del usuario de todos aquellos grupos donde se encuentre en el archivo */etc/group*.
- Borrar el directorio personal.

6.1 El super usuario

El super usuario es un usuario con ciertos privilegios y por supuesto más responsabilidades dentro del sistema, su clave de acceso es *root*. El super usuario puede ver y manipular todos los archivos del sistema, sin importar a qué usuario pertenezcan; es por eso que el usuario que posea este poder debe conocer bien los componentes del sistema. El super usuario tiene a su cargo la administración del sistema, y para ello cuenta con una contraseña que se espera sólo él conozca para seguridad de los demás usuarios y, por supuesto, del sistema mismo.

Es recomendable que el super usuario conozca sus privilegios y obligaciones, ya que tiene la capacidad de cambiar cualquier tipo de información en el sistema, desde la contraseña de algún otro usuario sin avisarle, borrar sus archivos, etc, hasta cambiar o borrar información en los archivos de configuración del sistema pudiendo causar incongruencias en ellos. Así, el super usuario debe usar su pequeño poder con cautela y siempre teniendo en cuenta que el sistema debe estar ejecutándose bajo las mejores condiciones.

Una característica importante del super usuario es que su UID es 0. Algunas de las tareas que sólo el super usuario puede realizar son [8]:

- Crear archivos de dispositivos.
- Distribuir los recursos del sistema usando prioridades y estableciendo límites.
- Configurar las interfaces de red.
- Apagar el sistema.
- Darle nombre al servidor.
- Poner la hora en el sistema.

El super usuario es el único usuario que puede configurar e interactuar directamente con el kernel de UNIX. Sin embargo existen otro tipo de pseudo-usuarios definidos por el sistema que lo pueden hacer [8]:

demonio *(daemon)*: Conocido como el **dueño no privilegiado del software del sistema**, el UID de este tipo de usuario es 1. Los archivos asociados a este UID pertenecen al sistema operativo o proveen a éste de servicios adicionales. Sin embargo no pueden ser de root, pues podrían poner en riesgo la seguridad del sistema mismo en caso de tener problemas al ejecutarse.

bin: También es conocido como **dueño del sistema de comandos**. A este pseudo-usuario pertenecen todos aquellos directorios que contienen los comandos del sistema. Éstos no le pertenecen a root por las mismas razones aducidas en el caso anterior.

sys: Este usuario es el **dueño del kernel y de la memoria** y se encarga de manejar el espacio de direcciones del kernel, la memoria física junto con las interacciones que hay de ésta con el espacio de memoria virtual en el área de swap. [2]

nadie *(nobody)*: Es el "dueño" de algunos archivos o directorios a los que se permite acceso público o por parte de un grupo de usuarios o bien que pueden ser accedidos o montados por computadoras remotas. Como ejemplo tenemos el área de acceso anónimo por FTP o los directorios o sistemas de archivos compartidos mediante NFS [3]. Algunos procesos utilizan el identificador *nobody* para proteger a root: si algún programa corre como si fuera ejecutado por *nobody* y tiene un error o es posible "explotar" algún hueco de seguridad en él, entonces el usuario malicioso sólo puede hacer tanto daño como lo permitan los privilegios de *nobody* que, por supuesto, se suponen mínimos.

Al crear un usuario es necesario decidir qué tipo de permisos tendrá, pues de esa forma el super usuario puede acotar perfectamente las limitaciones y obligaciones del usuario, delimitando también sus relaciones con el resto de los usuarios. Los permisos de acceso a archivos permiten hacer esto eficientemente. Dado que se aplican a cualquier tipo de archivos (incuyendo aquellos que contienen programas ejecutables o de configuración del sistema) es posible impedir el acceso de los usuarios a archivos importantes o a los pertenecientes a otros usuarios. En principio cada usuario puede escribir o borrar sólo los archivos en su directorio personal, no puede modificar los programas del sistema o alterar archivos de otros usuarios, posiblemente ni siquiera pueda leerlos, si así está especificado. Cada usuario es responsable de los archivos en su directorio personal y sólo de esos, y puede proveer acceso a ellos al resto de los usuarios o no, según sus intereses o necesidades.

Los permisos de acceso a los archivos se clasifican en:

- Lectura (r).
- Escritura (w).
- Ejecución (x).

Es posible cambiar los permisos con el comando *chmod* de la siguiente manera[8]:

[2] La información relevante a este respecto se encuentra en los archivos */dev/kmem, /dev/mem, /dev/drum, /dev/swap*

[3] El UID de nobody puede ser 1 o 2 dependiendo del sistema.

Permisos	Octal	Binario	Permisos	Octal	Binario
rwx	7	111	-wx	3	011
rw-	6	110	-w-	2	010
r-x	5	101	–x	1	001
r–	4	100	—	0	000

Tabla 6.3: Permisos en Linux.

6.2 Los Usuarios en el SisMo

El sistema SisMo monitorea la actividad de los usuarios y provee al administrador de la información siguiente:

Usuario: se refiere al nombre o identificador que tiene el usuario en el sistema. En este caso es el *login* del usuario.

Terminal (TTY): cada una de las terminales donde se encuentra trabajando el usuario esta descrita en este campo por *tty*. Un usuario puede tener más de una terminal al estar trabajando dentro del sistema.

Sistema Remoto: aquí se muestra el lugar desde donde se hizo la conexión al sistema. Si se esta trabajando directamente en la consola de la máquina sólo aparece el carácter '-'; si el usuario trabaja en la máquina mediante una conexión desde otra se muestra la dirección de esta última.

Procesos: este campo regularmente es mostrado con el comando *w*, bajo el campo de *WHAT*, el cual muestra el o los procesos que el usuario está ejecutando.

Sesiones: se muestra el número de sesiones que un usuario tiene abiertas.

Fecha de inicio de sesión: en este campo se encuentra información relacionada con la fecha de ingreso a la computadora de esa sesión.

Capítulo 7

La red en el kernel de Linux

Ya se ha hablado de redes de computadoras y de su gran y creciente importancia en la actualidad. Es por esto que la interfaz de red de cada computadora y la interacción entre los sistemas debe ser permamentemente vigilada y mantenida en codiciones óptimas.

Para un administrador de sistemas es de fundamental importancia la siguiente información [14]:

1. El nombre completo de la computadora.- Éste es elección del administrador, sin embargo debe de ser único.

2. La dirección IP de la computadora.- Esta dirección está constituida por cuatro números entre 0 y 255 (expresables en 8 bits en notación binaria), separados por puntos[1], los cuales identifican a cada interfaz de red. Hay que recordar que este identificador es propio de las redes con protocolos de la familia TCP/IP solamente y que identifican a una interfaz de red y no a una computadora, una sola máquina podría tener más de una dirección si es que tiene más de una tarjeta de red, por ejemplo.

3. La máscara de la subred. Se indica cuáles de los números de la dirección IP son comunes a todas las máquinas de la red. Una máscara de subred 129.79.123 indica que todas las máquinas de la subred tienen esos tres números y en ese orden en sus direcciones IP.

4. La dirección de difusión (*broadcast*).- Es posible enviar información a todas las máquinas de la subred así que se debe poder especificar una dirección genérica que las contenga a todas. Sabemos que todas las máquinas de la subred tienen algo en común: la dirección IP de cualquier máquina comienza con los mismos digitos, los de la máscara de subred, justamente. Así que para enviar información a todas las máquinas de la subred al mismo tiempo se envía a una dirección genérica que es la máscara de subred seguida de 1´s en la expresión binaria. En nuestro ejemplo la dirección de difusión sería 129.79.123.255 (el último número (255) es expresado como ocho unos consecutivos en binario).

[1] Por ejemplo: 129.79.123.156

5. La ruta por omisión para la computadora.- Un ruteador se encarga de asignar una ruta a cada paquete de datos que viaja por la red. Así que cada computadora debe enviar a un ruteador aquellos paquetes destinados a máquinas fuera de la subred. La dirección IP del ruteador es la *ruta por omisión*. Normalmente la dirección del ruteador es la dirección de la subred con un 254 al final; en nuestro ejemplo sería 129.79.123.254. Un ruteador tiene dos o más interfaces de red, cada una con una dirección diferente, que pertenecen a una subred.

6. El puerto de retroalimentación o *loopback*.- Es una interfaz que facilita la comunicación entre procesos diferentes en la misma máquina. Esta interfaz envía paquetes a sí misma [2]. (Normalmente se usa la dirección 127.0.0.1).

Al conectarse una máquina con otra, éstas usan su dirección IP para intercambiar información. Esta información esta contenida en paquetes, en los cuales el encabezado de IP contiene la dirección de la máquina fuente y la del destino. Es sumamente importante saber si esta información no se corrompió durante la transmisión y para esto se usa un *checksum* o *suma verificadora* que se encuentra en el paquete mismo. El tamaño de los paquetes varía dependiendo de las conexiones. Los paquetes pueden ser recibidos en un orden distinto al enviado, pues cada uno puede, en principio, ser enviado por distintas rutas con distintos retardos.

Con base en la clasificación de las redes de computadoras, hecha usando como criterio su alcance o el área cubierta por ellas, podemos clasificar a *Ethernet* como una red de área local o LAN. Ethernet es una de las redes locales más usuales para conectar equipos de cómputo en general y equipos Linux en particular; transmite a una tasa de $10,000,000$ de bits por segundo $10Mbps$ [12], aunque actualmente hay versiones de $100Mbps$ y está en desarrollo una versión de $1Gbps$. Existen muchas otras tecnologías pero Ethernet es la más conocida y usada entre las redes de área local.

Sobre la infraestructura provista por la red local se montan un conjunto de protocolos que, en su conjunto, brindan todos los servicios de comunicación de alto nivel que hacen la comunicación confiable y permiten a las aplicaciones de red abstraerse de los problemas de detección y corrección de errores y de señalización que se manejan en niveles más bajos. Los protocolos más usuales hoy en día son aquellos diseñados para la red ARPANET que se convirtió en nuestra actual Internet y que son conocidos comúnmente como protocolos TCP/IP como ya se había mencionado.

7.1 TCP/IP

Cada día más y más usuarios de las computadoras se están conectando con otros miles de usuarios y con servicios diversos, mantenidos todos por redes locales, conexiones a Internet, sitios *web* (red), etc. **TCP** es un conjunto de protocolos usados por muchos sistemas operativos (UNIX, MacOS, Windows, Windows NT, entre otros). Es el lenguaje de Internet y junto con **IP** [8] definen una interfaz uniforme que puede montarse en distintos tipos de hardware, garantizando el intercambio de datos de una manera segura.

[2] En Linux el comando *ifconfig* es usado para éste propósito

TCP es un protocolo orientado a conexión [3] que es confiable durante la transmisión, mantiene el control de flujo, evita la congestión de datos.

TCP es el encargado de romper el mensaje en paquetes para enviarlo [4], reensamblar éstos en orden correcto al recibirlos para reconstruir el mensaje original y enviar un acuse de recibo a la máquina que envió el mensaje, o bien solicitarle reenvío de algún paquete en caso de que éste se hubiere perdido. *IP* es el responsable de enviar y recibir los paquetes especificando el remitente y destinatario. Los paquetes de IP tienen su propio encabezado, así como los de TCP. De esa forma una vez que TCP envía un mensaje, lo hace primero a través de IP y cuando algún cliente envía un mensaje, es IP, quien primero lo recibe y se lo pasa a TCP. Los distintos campos en cada paquete son necesarios para poder comunicarse entre las computadoras que envían y reciben información.

TCP pone un encabezado al principio del paquete, el cual contiene la siguiente información: **número de puerto fuente y destino de TCP, número de secuencia (SEC), acuse de recibo (AR)**. El primero se refiere al número de puerto de la computadora fuente que será usada para establecer la comunicación, al igual que el número de puerto de la computadora destino. Cada servicio de comunicación (FTP, Telnet, http, etc.) posee un puerto específico. Después le sigue el número de secuencia (SEC) que sirve para poder determinar el número de paquete (el cual es único) que se está recibiendo y poder determinar el orden en que los paquetes deben ser pegados para reconstruir el mensaje. De esta manera es posible determinar también si algún paquete se perdió. Se hace una suma de verificación para poder determinar si el tamaño del mensaje es idéntico al enviado (o recibido) y posteriormente se encuentran los datos del mensaje. Una vez que TCP ha puesto su encabezado correspondiente se lo pasa a IP. Éste pega el encabezado de *tamaño del mensaje, el protocolo, la suma de verificación, la dirección IP de la computadora fuente,y la dirección IP de la computadora destino.* El primero se refiere al tamaño del mensaje en bits. El número del protocolo se refiere al protocolo de nivel superior a IP que envía el mensaje (p.ej. TCP). La suma de verificación es para poder verificar que lo que se ha recibido coincide con lo que se envió. Las direcciones de IP de las computadoras fuente y destino ocupan 32 bits y después de esa información se encuentran los datos del paquete. Describir en mayor detalle los campos de cada paquete es innecesario para los fines de este trabajo; el lector interesado en conocer más acerca de la construcción de paquetes de IP o de fragmentos de TCP puede consultar [3]. En la figura 7.1 se muestra esquemáticamente el contenido de los paquetes de TCP/IP.

Cuando se establece una conexión entre computadoras, éstas pasan por diferentes etapas, así que certificar que una máquina funciona correctamente en lo que a su interacción con otras se refiere, significa verificar el estado de todas sus conexiones. Las conexiones de TCP pueden encontrarse en alguno de los siguientes estados: establecido *ESTABLISHED*,

[3] Un protocolo orientado a conexión mantiene la conexión entre dos computadoras, aunque éstas no se encuentren permanentemente enviando o recibiendo datos. Un buen ejemplo de comunicación orientada a conexión es la que se lleva a cabo con una llamada telefónica. La conexión entre emisor y receptor es permanente hasta que alguno de los dos cuelga. En contraste, la comunicación no orientada a conexión (también llamada de **datagramas**) es análoga a un servicio de correo convencional, cada carta es enviada individualmente, el orden de recepción no necesariemente es el mismo que el de envío, algunas cartas pueden extraviarse.

[4] También conocidos como fragmentos.

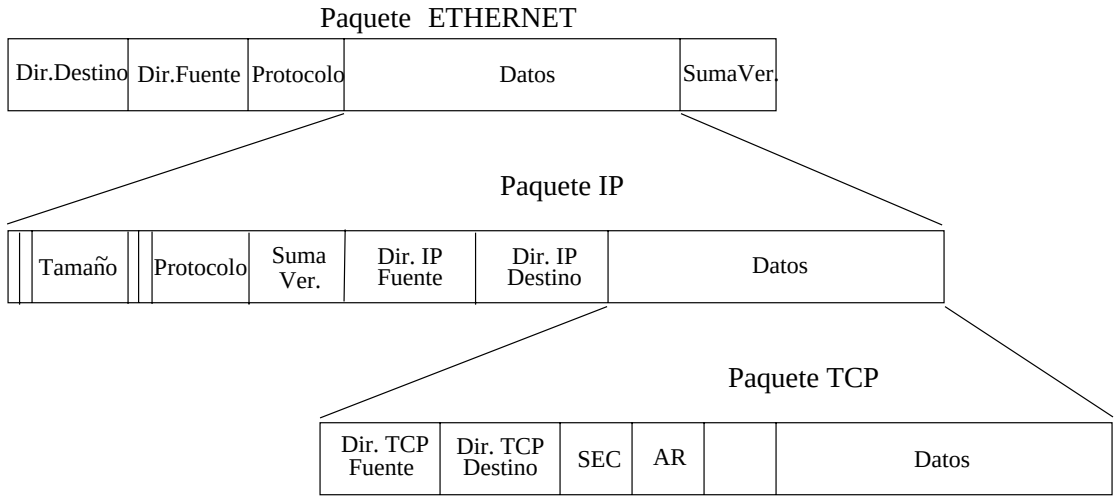

Figura 7.1: Paquetes de TCP/IP

escucha *LISTEN* o tiempo de espera *TIME_WAIT*; el primero se refiere a la conexión o conexiones que actualmente se encuentran activas; el segundo se refiere a las conexiones posibles, es decir, conexiones aún no establecidas para las que el demonio de TCP se encuentra en estado de escucha, esperando que se soliciten. El último se refiere a las conexiones que se encuentran en proceso de cerrarse.

Al enviar y recibir paquetes de datos puede haber errores. Los paquetes erróneos no deben exceder el 1% del total de paquetes [8]; si el porcentaje fuere mayor, probablemente el problema resida en la interfaz de red de la máquina o la conexión. Otra fuente de problemas son las colisiones; en una red del estilo de Ethernet, en la que existe un único canal de comunicación común a todas las computadoras, puede ocurrir que más de una computadora trate de enviar datos en un instante determinado, es decir, puede ocurrir que dos máquinas "hablen" al mismo tiempo en el canal; esto es una colisión, y por supuesto los datos enviados por las computadoras partícipes de una colisión se pierden y deben ser retransmitidos. Cuanto más intenso sea el tráfico en la red, mayor será el número de colisiones que ocurran, por lo que medir la frecuencia de éstas es útil para determinar que tan saturado se encuentra el canal.

Existen diversas herramientas de uso común para determinar el estado de la interfaz de red y/o de las máquinas en la red. Es usual, por ejemplo, utilizar el programa *ping* para determinar si una máquina está activa en la red o no, más precisamente, si responde o no a mensajes enviados a través de la red [5]. Otro programa usual para diagnosticar problemas con la conexión de red es *netstat*, con el que es posible obtener el estado de cada conexión y el estado general de cada interfaz de red.

[5] *ping* es una aplicación montada sobre el protocolo ICMP, al mismo nivel de TCP. El programa envía peticiones de "eco" a una máquina específica y si esta se encuentra activa (y tiene habilitado el protocolo ICMP), responde con el mismo paquete que recibió

7.2 La red en el SisMo

Dada la importancia de la conectividad de cada sistema en una red y de las labores que dependen de que dicha conectividad opere eficientementente, el sistema SisMo monitorea también el estado de las interfaces de red y de las conexiones. El reporte en este rubro contempla los siquientes aspectos:

Interfaz: se describe el tipo de interfaz que se está utilizando para la conexión en red.

Bytes recibidos: se refiere al número de bytes que han sido recibidos.

Paquetes recibidos: el número de paquetes que han sido recibidos.

Errores en los paquetes recibidos: número de errores ocurridos durante la recepción de los paquetes enviados a esa máquina específicamente.

Paquetes desechados en su recepción: hace referencia al número de paquetes que han sido desechados luego de haberse recibido. Esto ocurre cuando un paquete llega con errores que son detectados al recibirlo o bien cuando llegan paquetes duplicados.

Sobreflujo en los paquetes recibidos: número de paquetes perdidos debido a que la interfaz de red no tuvo la capacidad de almacenamiento o la velocidad suficiente para recibirlos.

Bytes transmitidos: número de bytes transmitidos por la máquina.

Paquetes transmitidos: número de paquetes transmitidos.

Errores en los paquetes transmitidos: hace referencia al número de errores que han ocurrido durante la transmisión de algún paquete.

Paquetes desechados en su transmisión: se refiere al número de paquetes descartados durante su transmisión.

Sobreflujo en los paquetes transmitidos: número de paquetes cuyo envío no pudo efectuarse en el primer intento, debido a insuficiencia de espacio para almacenarlos temporalmente mientras se envían.

Colisiones: se refiere al número de colisiones ocurridas en el último intento de envío de paquete.

/etc/exports. Es posible conocer en la máquina cliente el nombre, servidor y punto de montaje de cada sistema de archivos importado usando el comando *mount*.

Como se dijo anteriormente, al exportar y montar un sistema de archivos el kernel etiqueta éstos con ciertas banderas que se muestran en las tablas 8 y 8.

Bandera	**Descripción**
active=*lista*	Lista de las computadoras que pueden montar un sistema de archivos.
rw	El sistema de archivos es exportado con acceso exclusivamente de lectura sólo para aquellos que no se encuentren en la lista.
ro	El sistema de archivos es exportado con acceso exclusivamente de lectura para los clientes
root=*lista*	Solo las computadoras de la lista pueden acceder a los sistemas de archivos como *root*.

Tabla 8.1: Banderas en el servicio de exportación de archivos.

Bandera	**Descripción**
rw	Monta el sistema de archivos en modo de lectura-ejecución.
ro	Monta el sistema de archivos en modo de sólo lectura
bg	Si el montado de archivos falla (el servidor no responde) lo mantienen en background y continúa con otra petición de montaje
hard	Si el servidor se cae, las operaciones se bloquean hasta que el servidor se recupera.
soft	Si el servidor se cae, el acceso falla y regresa un error
intr	Permite que los usuarios interrumpan operaciones bloqueadas (y obligarlas a que regresen un error)
nointr	No mantiene las interrupciones de los usuarios
retrans=n	Especifica el número de veces que ha de repetir la petición antes de regresar un error
rsize=n	Especifica el tamaño del buffer leído en n *bytes*
wsize=n	Especifica el tamaño del buffer escrito en n *bytes*
tcp	Selecciona la vía de transporte.

Tabla 8.2: Banderas en el servicio de montaje de archivos.

A través de NFS los directorios pueden accederse como si fueran locales. Este tipo de comunicación es clasificada como de arquitectura *cliente-servidor* dado que consiste de:

- Un programa cliente.
- Un programa servidor.

- Un protocolo de comunicación entre los dos anteriores.

El programa servidor es el que hace posible la exportación de sistemas de archivos, es decir permite a otras máquinas el acceso a sistemas de archivos locales. A los sistemas de archivos que pueden ser accedidos por computadoras remotas se les denomina *compartidos*. El programa cliente es el que tiene acceso a los archivos compartidos por otra máquina y hace ver este acceso como si fuera local. Cuando un sistema de archivos está montado, éste se integra al árbol de directorios [14]. El protocolo NFS se encarga de la comunicación entre el cliente y el servidor, de tal forma que el cliente puede acceder a árboles de directorios remotos en un *punto de montaje* en el sistema local. Un punto de montaje es un subdirectorio o directorio vacío, y es a través de éste que se puede acceder a cierto sistema de archivos, es por así decirlo, un punto de refrencia desde donde es posible acceder a los sistemas de archivos remotos. Para poder montar un sistema de archivos de un servidor, el usuario necesita una cuenta en la máquina donde se encuentra el sistema de archivos. De esta forma, el cliente pasa el UID y el GID del proceso solicitante, así como el nombre del sistema de archivos requerido al servidor de NFS y éste se encarga de validar la petición antes de atenderla. El protocolo de montaje puede asignar ciertos privilegios remotos a un número restringido de clientes usando *export*. A continuación algunos de los demonios más conocidos de NFS:

nfsd: se encarga de atender peticiones de acceso a los sistemas de archivos locales por parte de clientes remotos.

biod: se encarga de mantener las peticiones de entrada y salida de los procesos clientes.

rpc.mountd: mantiene el montaje de sistemas remotos.

rpc.lockd: maneja los archivos de bloqueo en el servidor de NFS y en las máquinas cliente.

rpc.statd: maneja las caidas de sistema o de red y recupera los servicios tanto para los clientes como para el servidor.

portmap: facilita la conexión de una máquina local y del servidor remoto [2].

NFS ofrece muchas ventajas, permite compartir recursos valiosos (información y espacio de almacenamiento) haciendo transparente para el usuario que algunos de estos recursos no son locales. El desempeño de NFS ha ido mejorando a lo largo del tiempo y la cantidad de almacenamiento ha ido creciendo, por lo que hoy en día soporta varios terabytes en cada sistema de archivos y a cientos de usuarios. Además es fácil de administrar y facilita otras tareas de administración típicas, como el respaldo de sistemas de archivos remotos [8]. La desventaja típica de NFS es que puede constituir un hueco de seguridad, una puerta de entrada a usuarios no autorizados o un medio para adquirir privilegios de acceso en un sistema.

[2]Este demonio no forma parte propiamente de NFS, sin embargo mejora notablemente el desempeño.

8.1 El automontaje

Montar un sistema de archivos en tiempo de inicialización de la máquina *(boot time)*, junto con una lista pre-compilada de equipos que puedan acceder a éste, puede llegar a ser una tarea difícil y costosa. Por tal motivo se creó el **automontaje**, el cual permite de manera eficiente, montar los sistemas de archivos cuando éstos son requeridos, y desmontarlos cuando ya no lo sean. Esto minimiza el número de puntos de montaje activos [8] siendo esta tarea transparente para los usuarios del sistema. El dispositivo de manejo del automontaje utiliza un archivo de mapa (*map file*), que hace posible decidir qué sistema de archivos es posible montar. El demonio de **automount** es el que se encarga de escuchar las peticiones en los directorios del NFS. Cuando llega una solicitud, el manejador de automontaje atiende esa petición dejándola en el área de trabajo *scratch*. Y la petición ha sido satisfecha.

Existen diversos tipos de mapas, como *indirectos, directos y maestros*.

Mapas Indirectos: son los más usados y pueden automontar varios sistemas de archivos sobre un directorio común. Se pueden usar en directorios locales, cuyos subdirectorios se montan desde equipos remotos. La ruta del directorio se especifica en el mapa maestro. En el siguiente ejemplo de un mapa indirecto, se muestran todas las entradas del directorio `/home` [12]:

```
piegrande -rw,intr stomp:/home/piegrande
dracula   -rw,intr castle:/home/dracula
quasimodo -rw,intr church:/home/quasimodo
```

Mapas Directos: lista el sistema de archivos aunque las entradas no tengan el mismo prefijo, es decir aunque no compartan el mismo directorio padre [3]. El siguiente ejemplo lo muestra:

```
/usr/src
/cs/tools
```

Mapas Maestro: lista los mapas indirectos y directos que puede automontar.

8.2 El NFS en el SisMo

Debido a que el NFS provee acceso remoto para compartir sistemas de archivos a través de la red, el sistema SisMo debe encargarse de monitorear aquellos que han sido importados, exportados y montados en alguna máquina. Es importante resaltar que el importar es un caso particular del montaje, cuando el sistema de archivos es remoto. En general se obtiene la siguiente información respecto a los sistemas de archivos relacionados con NFS:

Importados: se refiere al número de sistema de archivos remotos, montados localmente.

[3] El comando `ls` tiene la misma estructura.

Exportados: se refiere al número de sistema de archivos que han sido exportados.

Montados: se refiere al número de sistemas de archivos remotos o locales a los cuales se tiene acceso a través del sistema operativo.

Capítulo 9

La impresora en el kernel de Linux

Si bien las impresoras, como muchos otros dispositivos, han sufrido cambios tecnológicos que se reflejan en características que facilitan enormemente su manejo y optimizan sus funciones, la característica esencial de las impresoras, que las distingue del resto de los dispositivos, sigue y seguirá estando presente, es intrínseca a su funcionamiento: la imposibilidad de uso concurrente. El sistema operativo puede administrar la memoria o el disco duro de tal forma que muchos procesos puedan acceder a ellos por periodos de tiempo que se traslapan, pero no es posible hacer esto con las impresoras: no es aceptable que en una página aparezcan impresas algunas líneas de texto del archivo de un usuario y otras de otro archivo y/o de otro usuario. El proceso de impresión sigue siendo, esencialmente, un proceso por "lotes" (*batch processing*) y la complejidad de su manejo se ha incrementado por el hecho de que, en los entornos actuales típicos, una sola impresora es compartida por varias computadoras a través de la red. Proveer acceso a los servicios de impresión constituye así, una parte importante de la administración de cualquier red local.

Para proveer de servicios de impresión la computadora debe contar con el hardware y software adecuados. En particular debe contar con un **servidor de impresión** (*spooler*), que almacena los trabajos para después procesarlos. El spooler forma las peticiones de impresión en una cola, de donde las toma para enviarlas a la impresora cuando está desocupada. Los trabajos se van imprimiendo en el *background*. El servidor de impresión en Linux es, usualmente, **lpd**.

9.1 El demonio de `lpd`

El acceso a las impresoras en muchos sistemas Unix y en Linux en particular es controlado por el demonio `lpd` [1] y se inicializa cuando tiene lugar la carga del sistema (*boot*). Se encarga de aceptar las distintas peticiones de impresión de los usuarios, las procesa y una vez aceptadas las envia a la impresora cuando ésta está disponible. A través del comando `lpr` el usuario manda a imprimir un trabajo a la impresora, el cual es enviado también a `lpd`. `lpr` y `lpd` se comunican a través de un socket [2]. `lpr` verifica la línea de comando, para saber a qué impresora se está encargando el trabajo o si se sumistrará a la impresora por omisión,

[1] Se puede localizar en /usr/sbin

[2] El socket `/dev/printer`

entre otras cosas. Una vez que `lpr` conoce el destino, verifica el archivo `etc/printcap` el cual le indica a `lpr` en cual subdirectorio (cola de impresión) del directorio de *spool* (`/var/spool/lpd/nombre_de_la_impresora`) debe colocar el archivo a imprimir, para que de allí sea tomado y enviado a la impresora.

Cuando accede al directorio *spool*, `lpr` crea dos archivos por cada trabajo de impresión:

1. Archivo de control *control file (cfxxx)* o archivo temporal *temporary file (tfxxx)*.

2. Archivo de datos *data file (dfxxx)*.

El primero contiene información respecto al trabajo a imprimirse, como referencias a éste y el nombre del dueño del trabajo en cuestión, y el segundo archivo contiene los datos que serán impresos. Una vez que `lpr` notifica a `lpd` que tienen algún trabajo, éste consulta el archivo `printcap`, el cual decide si el trabajo es local o remoto. [8].

Los trabajos de impresión son atendidos como en una cola convencional: el primero que entra es el primero que sale (*Firs-In Frst-Out o FIFO*)[3].

De esta forma, una vez que `lpd` comienza, lee el archivo `/etc/printcap`, en el cual está definido el sistema de impresión. Cada vez que se imprime algún trabajo, éste tiene que pasar por el directorio *spool* y `lpd` vuelve al estado previo, es decir a esperar alguna petición de impresión. `lpr` es el único programa que se encarga de formar en la cola los archivos a imprimir.

Los comandos involucrados en labores de impresión se muestran a continuación:

Comando	**Ubicación**	**Descripción**
lpq	/usr/bin	Muestra la cola de impresión y su estatus.
lpr	/usr/bin	Encola un trabajo para su impresión.
lprm	/usr/bin	Cancela la impresión de un trabajo.
lpc	/usr/bin	Controla la cola de impresión y a la impresora.
lptest	/usr/bin	Genera un patrón de prueba en ASCII.
printtool	/usr/bin	Configura el sistema de impresión.
lptcontrol	/usr/bin	Configura el puerto paralelo para la impresión.

Tabla 9.1: Comandos para la impresión en Linux.

9.2 El administrador de los cambios `lpc`

`lpc` puede hacer algunos cambios en lo referente a la impresión, no puede usarse a través de la red, y se utiliza en modo interactivo.

Las funciones de `lpc` son [8]:

[3]El administrador del sistema puede modificar este esquema con `lpc`, en la agenda de impresión de trabajos individuales.

- Habilitar y deshabilitar la cola de alguna impresora, a través de los siguientes comandos:
  - `enable` *nombre_impresora*
  - `disable` *nombre_impresora*
- Habilitar y deshabilitar la impresión de alguna impresora, con los comandos:
  - `start` *nombre_impresora*
  - `stop` *nombre_impresora*
- Qutar todos los trabajos de la cola de impresión, por medio del comando:
  - `abort` *nombre_impresora*

  Este comando vuelve a su estado de impresión de trabajos, cuando la impresora es rehabilitada.
- Mover algún trabajo al tope de la cola de impresión, haciendo uso del comando:
  - `topq` *nombre_impresora id_trabajo*

  El cual mueve el trabajo con identificador $<$ *id_trabajo* $>$ al frente de la cola asociada a la impresora $<$ *nombre_impresora* $>$.
- Detener, iniciar o reiniciar el demonio `lpd`. Por ejemplo la acción de reiniciar se logra a través de:
  - `restart` *nombre_impresora*
- Obtener información acerca del estado de la impresora a través del comando:
  - `status` *nombre_impresora*

9.3 El archivo `printcap`

Este archivo contiene información acerca de las impresoras locales y remotas, las cuales deben estar especificadas en este archivo antes de que puedan realizar cualquier tipo de tarea. Se encuentra en `/etc/printcap`. Contiene el nombre de la impresora, separado con barras verticales y le sigue su conjunto de configuración separado por dos puntos `':'`. Las opciones son `xx`, `xx=cadena`, `xx=número`, en donde `xx` representan dos carácteres para el nombre y los números asignados a ésta[4] [8]. El archivo especifica:

- Nombre de la impresora.
- Abreviación del nombre de la impresora.
- El nombre/número del cuarto de impresora.

[4]EN caso de que ningún valor sea asignado, por omisión toma el valor de "verdadero"

- Descripción.

Para poder imprimir cualquier trabajo, el archivo `printcap` debe conocer cierta información relacionada. En la descripción se pueden localizar conjuntos de configuración como el nombre del dispositivo (se puede localizar en las líneas seguidas de `:lp` *device name*); el directorio de almacenamiento de trabajos (formados en cola) `:sd` *(spool directory)*; el archivo de la bitácora de errores `:lf` (*log file*) que tipo de archivo comúnmente conocido como "log"; seguido de la especificación de la conexión de las impresoras de lectura-escritura `:rw`; el tamaño máximo del archivo `:mx`; el nombre de la máquina remota `:rm` y el nombre de la impresora remota `:rp`. El archivo de la bitácora de errores **lf**, guarda los errores que se generan durante la impresión.

La variable **sd**, directorio de almacenamiento de trabajos formados en la cola *(spool directory)*, debe estar en el mismo directorio padre usualmente conocido como `/var/spool/lpd` y tiene el mismo nombre que la impresora. Forma a los archivos que permanecen almacenados hasta poder transmitirlos a la impresora. Este a su vez tiene información acerca del *estado* (*status*), información que es recibida por `lpd` y vista por `lpq`[5].

El nombre del dispositivo **lp** especifica la impresora local. Su nombre se encuentra regularmente en el directorio `/dev`.

La variable **ac**, archivo de contabilidad, *(accounting file)*, mantiene registros de los recursos usados por los trabajos individuales. Estos registros sirven para regular y calcular cambios en los recursos.

La variable **mx** especifica el tamaño máximo del archivo *(max size limits)*, es decir, la cantidad de datos que pueden ser fomados al mismo tiempo.

A continuación se muestra una tabla con algunas de las variables más usadas en el archivo `printcap` [8]:

Nombre	**Tipo**	**Descripción**
sd	cadena	Directorio de almacenamiento de trabajos (encolados).
lf	cadena	Archivo de bitácora de errores.
lp	cadena	Nombre del dispositivo.
af	cadena	Archivo de contabilidad.
rm	cadena	Nombre de la máquina remota.
rp	cadena	Nombre de la impresora remota.
of	cadena	Filtro para la salida.
if	cadena	Filtro para la entrada.
mx	número	Tamaño máximo del archivo.

Tabla 9.2: Variables en la configuración del archivo `printcap`.

Para obtener información acerca de un trabajo en impresión, el sistema permite ejecutar el comando `lpstat`, el cual da información acerca del estatus del archivo. La siguiente tabla muestra algunas de las banderas con las cuales puede operar [8]:

[5]El archivo `lock` no permite invocaciones múltiples a `lpd`.

Nombre	Descripción
-r	Muestra el estado del demonio `lpsched` [6].
-d	Muestra el destino por omisión.
-u	Muestra el estatus de los trabajos solicitados por el usuario.
-p	Muestra el estado de la impresora.
-s	Muestra un resumen acerca de la información del estado.
-t	Muestra toda la información de estado.

Tabla 9.3: Banderas del comando `lpstat`.

9.4 La impresora en el SisMo

En el sistema SisMo se monitorea el estado general de las impresoras, con vistas a obtener información útil que permita diagnosticar correctamente el funcionamiento incorrecto del servicio de impresión y tomar las medidas correctivas pertinentes. La información obtenida por el sistema de monitoreo respecto a los trabajos de impresión es la siguiente:

Activos: se refiere al estado del trabajo, es decir si se encuentra imprimiéndose o no.

Dueño: hace referencia a la persona que es dueña del trabajo en espera o actualmente en impresión.

Trabajo: identificador del trabajo de impresión.

Archivo: nombre del archivo que se requiere imprimir.

Tamaño: tamaño del trabajo que requiere imprimirse.

Capítulo 10

Servidor del SisMo

En este capítulo se describen las dos primeras capas de la arquitectura del SisMo, (véase la Sección 2.2 en la página 16): la capa que almacena los datos que se reciben de los clientes y la capa intermedia, que funge como servidor y controla:

- la comunicación entre los clientes y las bases de datos,
- la consistencia de los datos almacenados, y
- la interfaz con el administrador del sistema.

En realidad la capa base es muy sencilla, ya que solo realiza las labores concernientes al almacenamiento de los clientes del SisMo. El análisis de los datos y su validación son realizados por la capa intermedia. En la siguiente sección se describe el modelo de la base de datos, y el sistema que analiza y agrega datos a dicha base.

10.1 Capa base: Modelo de la base de datos

La tabla más importante, es la que especifica la relación entre los equipos y los programas cliente que se ejecutan en cada uno de ellos. Es decir, una tabla de la base de datos, *kid*, guarda la información relevante para saber **qué cliente** se conectó al servidor, desde **qué equipo** y exactamente a **qué hora**. De esta forma, cuando el servidor tenga que extraer información de ese equipo y de ese cliente particular para mostrársela al administrador, sólo tiene que checar esta tabla y obtener las ligas (identificadores únicos) a la información extensa (que se encuentra en otras tablas) en un periodo dado de tiempo.

Por supuesto, la base de datos contempla una tabla para almacenar la información de cada tipo de cliente. Es decir, hay una tabla que almacena la información de los usuarios, otra para los procesos, para el procesador, para el sistema de archivos, etc. La discusión de éstas es poco interesante y por cuestiones de espacio no se abordará.

Lo que sí resulta interesante en el diseño de las bases de datos es su escalabilidad. Como ya se mencionó al inicio de este trabajo, una ventaja de un sistema como SisMo es que puede ser escalado para monitorear a más de un sistema Linux. Para lograr tal objetivo, las bases de datos deben soportar la interacción de varios administradores y de varios equipos. Así, se integraron tres tablas más a la base de datos:

hosts: Esta tabla guarda la dirección IP, el *hostname* (nombre del equipo más su dominio) y asigna un identificador único a cada equipo que se agrega al SisMo. Como se explicará más adelante, cualquier administrador ya registrado puede agregar nuevos equipos al sistema.

admins: La tabla de administradores guarda un *login*, el nombre y una contraseña (*password*) para cada uno de los administradores registrados en el sistema, además de un identificador único que les es asignado al momento de ser agregados a la tabla. Un administrador, además de poder monitorear varios equipos, también los puede dar de alta (registrarlos) en el sistema, y puede añadir también nuevos administradores. Más adelante se explicará cuál es la lógica en este diseño y también cómo puede ser modificado para adaptarse a diversos ambientes.

admin_host: Esta tabla mantiene una relación *N:N* (muchos a muchos) entre las dos tablas anteriores, de tal suerte que un administrador pueda estar relacionado (*administra a*) con varios equipos y viceversa, un equipo pueda estar relacionado (*es administrado por*) muchos administradores. Esto permite modelar de manera exacta la realidad, ya que, dependiendo del tamaño del sitio, pueden existir muchos equipos Linux que deseen ser monitoreados y puede existir un *equipo de administración*, en lugar de un administrador único. Más aún, pueden existir personas encargadas de distintos equipos, aún dentro de una misma organización. Ejemplo de esto son la mayoría de las redes universitarias, donde por su tamaño y naturaleza de trabajo, es usual encontrar redes administrativas, de investigación y de docencia, todas ellas separadas y con equipos de administración (y políticas de administración) distintas.

No se describe a detalle la definición de las tablas de la base de datos, porque su diseño es muy sencillo y distraería la atención de nuestra labor esencial. Algo que sí resulta vital y que se discutirá a continuación, son algunas de las herramientas que influenciaron definitivamente la programación del SisMo.

10.1.1 PLT Scheme

La capa intermedia, que se menciona más adelante en este mismo capítulo, está escrita en su totalidad en el lenguaje de programación Scheme; para ser exactos en la implementación de Scheme del *Programming Languages Team* (PLT). Esta decisión se tomó después de evaluar detalladamente diversas opciones, entre otras Perl, PHP y Java. Todas ellas tienen sus puntos a favor y sus puntos en contra, pero para efectos prácticos eran casi equivalentes. PLT Scheme fue el mejor cuando se considera su soporte para hacer programación *interactiva* en el Web [9], un concepto relativamente nuevo y que, como se verá más adelante, facilita enormemente el tipo de programación que es necesaria en la capa intermedia.

El enfoque del PLT para hacer programación interactiva se basa en la creación de un servidor de Web que corre sobre una máquina virtual de PLT Scheme y un esquema para evaluación de CGIs[1] (*Common Gateway Interface*) similar al que usan los demás servidores de Web, por ejemplo Apache (que es el líder indiscutible en este apartado a nivel mundial), pero le agregaron una estructura de control más, utilizando continuaciones (*continuations*)

de Scheme para permitir "suspender" momentáneamente la ejecución de un CGI mientras obtiene más información del usuario.

Esta característica de suspender y continuar la ejecución de un CGI es única y es de gran ayuda porque se evita la necesidad de hacer *marshaling/unmarshaling*. Es decir, si la ejecución de una cierta tarea requiere (como ocurre en la capa intermedia del SisMo) obtener datos en dos o tres pasos distintos, la forma usual de hacer eso con CGIs es:

1. El usuario introduce un cierto conjunto de datos por medio de una forma. Esos datos tienen que ser procesados y *guardados* de alguna manera: en una base de datos, en un archivo temporal, etc. Esto se conoce como *marshaling*. El resultado de este primer CGI es otra página con una nueva forma que pide el siguiente conjunto de datos al usuario.

2. Cuando el usuario introduce el segundo conjunto, los datos se vuelven a procesar y si entre los dos conjuntos (el recién leído y el anterior guardado en un archivo o en una base de datos) se puede completar la operación, entonces se hace el proceso inverso para sacar los datos del área de almacenamiento temporal (*unmarshaling*) y se completa la operación. Si se requieren más datos, se guarda el segundo conjunto de datos, de la misma manera que el primero, y se piden más y este proceso se repite hasta que se tenga toda la información para llevar a cabo la tarea.

Claramente, hacer este tipo de procesamiento es costoso y además propenso a errores. Con el uso de continuaciones en el servidor de Web de PLT Scheme y con su modelo de ejecución de CGIs, estos problemas se resuelven sin necesidad de hacer *marshaling/unmarshaling*. La manera en que se resuelve un problema como el expuesto anteriormente con este esquema es la siguiente:

1. El usuario introduce un cierto conjunto de datos por medio de una forma. Esos datos son procesados y se quedan en el lugar dónde llegaron por primera vez: el *heap* de la máquina virtual de PLT Scheme. El resultado de la ejecución de este CGI es una forma que pide el siguiente conjunto de datos y que dirige la ejecución de esos datos a una *continuación*, de este mismo CGI,

2. Cuando el usuario introduce el segundo conjunto de datos, se regresa a la ejecución del CGI anterior, i.e. a la *continuación* del CGI anterior con los datos nuevos (del segundo conjunto) y los datos viejos (del primer conjunto) ¡están todos ahí; en la memoria del programa en ejecución!. Claro que este proceso se puede repetir cuantas veces sea necesario, de la misma forma que en los esquemas usuales, y siempre usando continuaciones, por lo que nunca es necesario almacenar datos intermedios.

Debe ser claro que el nivel que este esquema ofrece es adecuado para capturar las interacciones necesarias en nuestra capa intermedia, como se verá en la siguiente sección.

[1] *Ejecutar un CGI* es la forma usual (no la única) de crear contenido dinámico para mostrar en una página de Web.

10.2 La capa intermedia

Como ya se mencionó, la capa intermedia se divide en dos partes a su vez: un **servidor** que recibe las comunicaciones de todos los programas cliente, y la otra mitad, el **manejador de administradores**, que puede ser visto como un cliente especial de la primera parte. Esto es, la función del servidor es: almacenar y extraer información de las bases de datos a petición de los clientes, mientras que el manejador de administradores se encarga de procesar los comandos que los administradores de los sistemas monitoreados ingresen (por medio del Web), envía las peticiones adecuadas al servidor, procesa los resultados y los envía de regreso (vía Web) al *browser* del administrador.

A continuación se presenta algunas características relevantes del servidor, seguido del manejador de administradores.

10.2.1 Servidor del SisMo

El servidor es un demonio que siempre está escuchando a la espera de peticiones de conexión por el puerto 8989[2]. Cabe aclarar que el servidor reconoce un único protocolo de comunicación, a diferencia de lo que se muestra en la Figura 2.2, donde aparecen un protocolo *A* y un protocolo *B*. En la figura se manejan dos tipos de protocolos para hacer obvia la diferencia entre los dos tipos de clientes que atiende nuestro servidor; en la práctica, es más sencillo hacer un protocolo y simplemente restringir a los clientes a usar el sub-conjunto de dicho protocolo que se apegue de mejor forma a sus necesidades.

Handshake

Por simplicidad, cuando las máquinas objetivo se conectan al puerto donde está escuchando el servidor, simplemente envían una cadena de datos que el servidor analizará y pueden o no esperar una respuesta antes de cerrar el canal de comunicación. En particular, los clientes que están instalados en los equipos monitoreados (los programas en Perl) interactúan de manera más compleja con el servidor que los clientes de Web.

Los clientes en los equipos, siguen el siguiente proceso para comunicarse y enviar datos al servidor:

cliente: `"hello:kid:host:ts"`

Donde, `kid` es un identificador único del tipo de cliente que se está conectando. Estos identificadores están integrados en los programas cliente. Así por ejemplo, el `kid` del cliente que envía datos sobre la memoria es `mem`; `host` es el nombre del equipo y `ts` es el tiempo en que se ejecutó el cliente (*timestamp*).

SisMo: Si los datos recibidos concuerdan con los datos de un host monitoreado, entonces el servidor contesta: `olleh`.

[2] 8989 es tan bueno como cualquier otro puerto no usado en la máquina anfitriona del SisMo, se ha fijado en una variable en el programa, pero podría convertirse facilmente en un parámetro que el administrador general del SisMo pudiese configurar a placer.

cliente: Después de recibir un `olleh`, comienza a enviar todos los datos. Los datos son cadenas separadas por dos puntos (':'). Esos datos son siempre los mismos para cada tipo de `kid`. Por ejemplo, el cliente que analiza el sistema de archivos (con `kid`, `fs`) siempre envía cadenas con seis datos: sistema de archivos, número de bloques (totales), número de bloques usados, número de bloques libres, el porcentaje de uso y el punto de montaje (directorio de montaje) en el sistema de archivos del host.

cliente: Cuando el cliente termina de enviar los datos, envía un punto (`.`) para indicar que ha terminado.

SisMo: Cuando recibe el punto, envía `bye` al cliente y cierra la conexión.

El cliente que utiliza el CGI para comunicarse con el SisMo es mucho más sencillo y su esquema de comunicación es el siguiente:

cliente: Envía una cadena cuyo primer elemento es el nombre de un proceso que el servidor debe realizar y cuyos siguientes elementos son los parámetros necesarios para que el servidor lleve al cabo ese proceso. Por ejemplo, el cliente puede enviar: `addhost:UID:IP:Hostname`, que le pide al SisMo que añada un equipo a su lista de equipos que son monitoreados, `UID` es el identificador del administrador que pide la acción, `IP` es la dirección IP del host a monitorear y `Hostname` el nombre (y dominio) del equipo.

SisMo: El servidor recibe la cadena, analiza los parámetros y si puede (y está autorizado) a realizar la acción, la hace y regresa la información solicitada o un mensaje indicando el resultado.

Por otro lado, la información que el cliente de Web puede recibir del SisMo es más compleja y para reducir los tiempos de análisis de los datos, envía información estructurada. Así, cuando el cliente de Web pide la lista de todos los equipos monitoreados por un administrador, el SisMo regresa una lista (la estructura de datos básica de Scheme) que contiene una sub-lista (otra lista) con la información de cada equipo que concuerde con la búsqueda.

10.3 Capa externa: Interfaz para el administrador

La interfaz que utilizarán los administradores de sistemas que utilicen el SisMo, como ya se había mencionado, será el servidor de Web de PLT Scheme, por medio de un CGI que hace varias cosas:

1. Le presenta una interfaz uniforme e independiente de la plataforma a los administradores. Las páginas que se usan son lo suficientemente sencillas como para verlas con prácticamente cualquier browser.

2. Implementa un cliente del SisMo que recibe las instrucciones del administrator, las pasa al lenguaje del protocolo acordado en secciones anteriores de este mismo capítulo, las comunica al demonio del SisMo y recibe los resultados para mostrarlos nuevamente.

10.3.1 Interfaz de Web

La interfaz de Web tiene dos partes, una muy breve que es la parte estática, que simplemente da la bienvenida al administrador y le ofrece una opción para *iniciar una sesión*, -véase la Figura 10.1- y una parte dinámica, que se obtiene desde el momento de haber iniciado una sesión exitosamente.

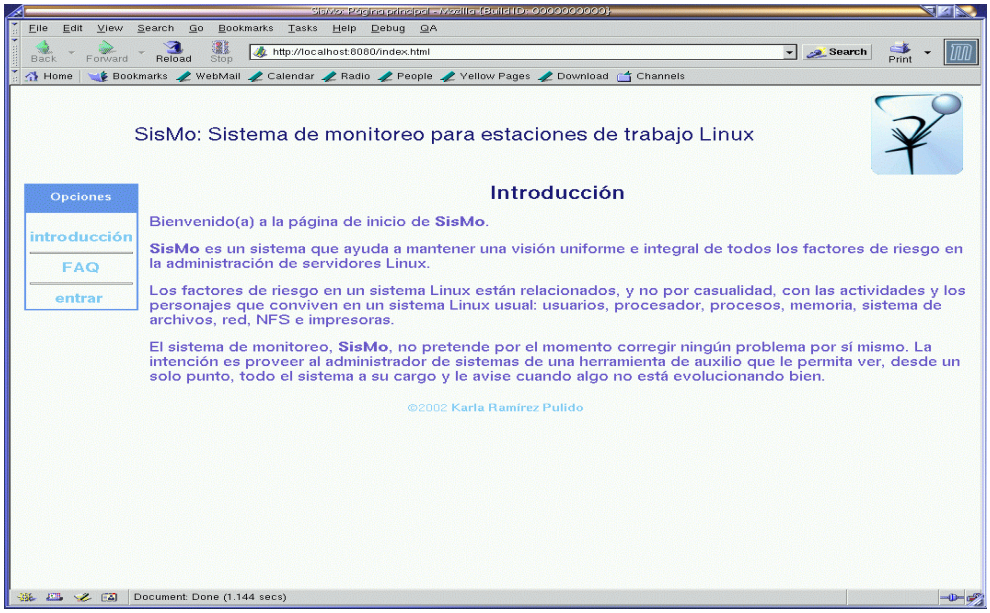

Figura 10.1: Inicio de sesión del SisMo

Es importante aclarar, que la parte dinámica es totalmente controlada de la misma forma: por medio de continuaciones dentro del servidor de Web. Entonces, desde que se entra al sistema, hasta que se sale de él (*logout*), toda la interacción con las páginas del sismo se hace a través de continuaciones. Esto hace que la mayoría de las operaciones sean eficientes, excepto quizás las que tienen que generar gráficas, porque involucran muchas acciones: pedirle al demonio del SisMo los datos (potencialmente muchos), darles el formato apropiado, llamar a un programa en *C*, que genera la gráfica y finalmente mostrar la página con el resultado (que es también una continuación). En la secuencia de gráficas 10.2, 10.3, 10.4, 10.5, 10.6, 10.7, y 10.8, se muestra la secuencia que el administrador sigue para ver el comportamiento del procesador del equipo que está monitoreando en el ejemplo: *lambda.fciencias.unam.mx*.

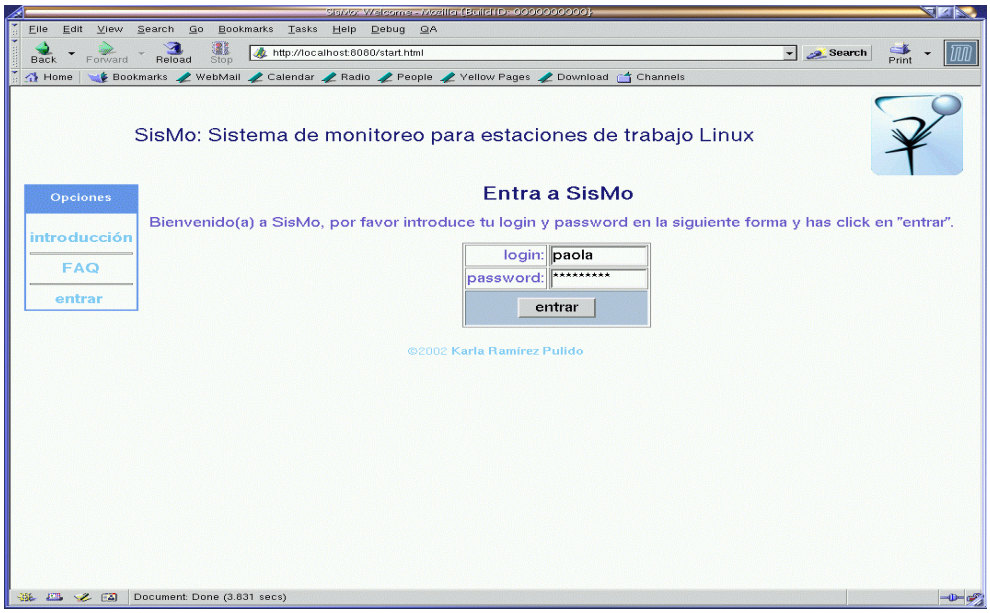

Figura 10.2: Forma para entrar al SisMo.

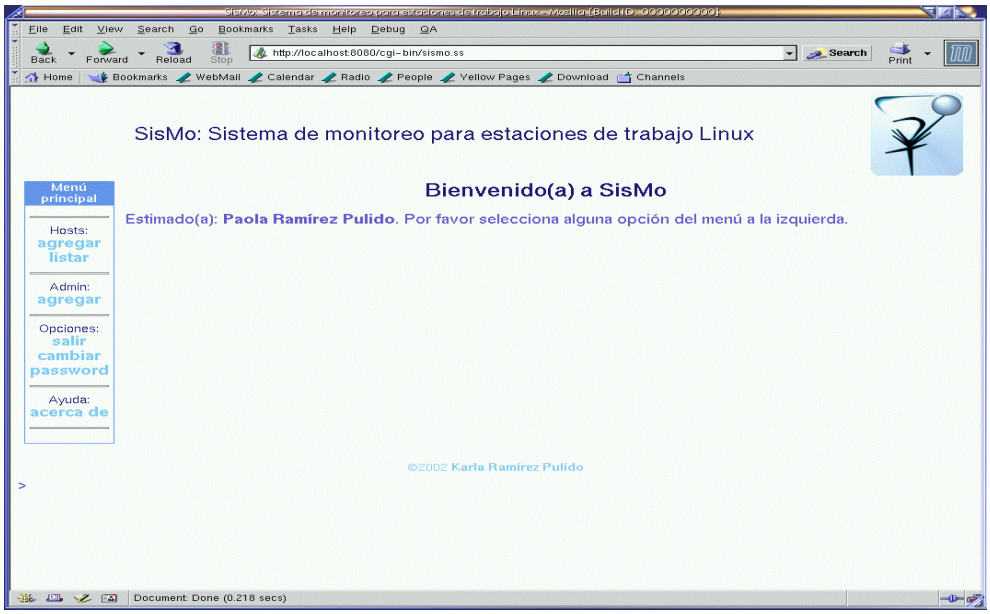

Figura 10.3: Después de entrar al SisMo

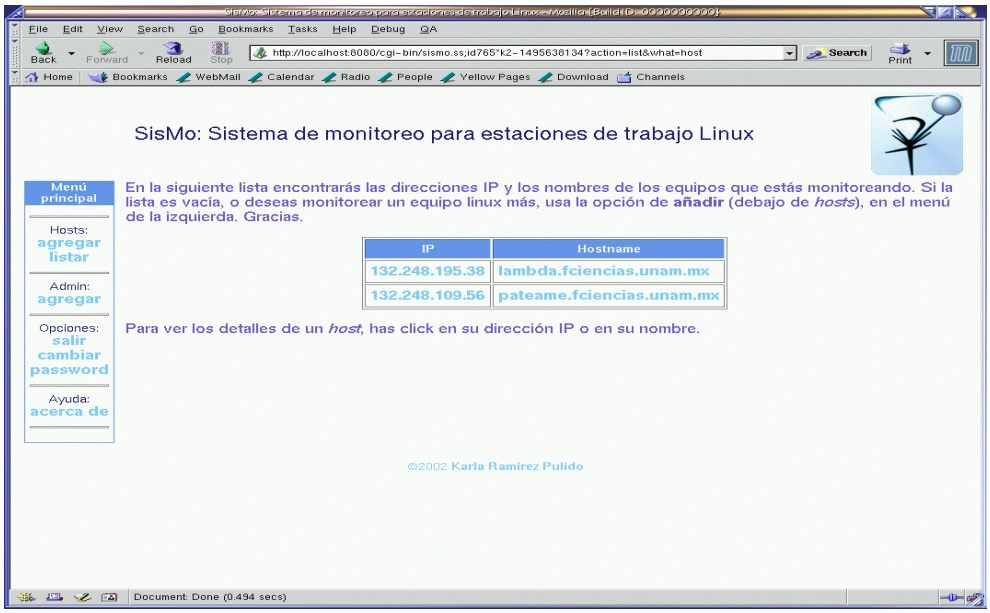

Figura 10.4: Después de seleccionar la opción listar en el menú principal del SisMo.

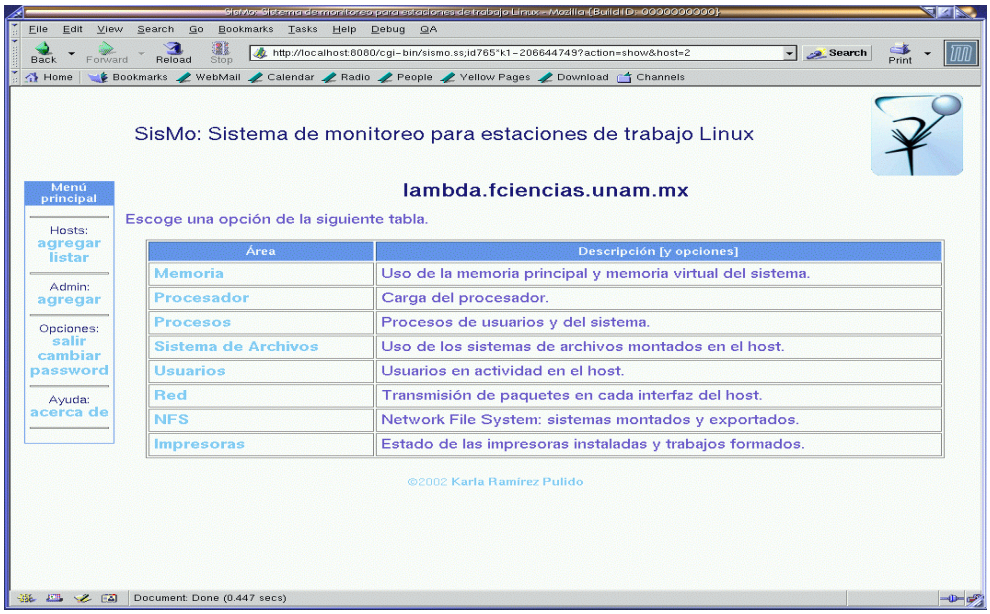

Figura 10.5: Las opciones a seleccionar para el equipo *lambda.fciencias.unam.mx*.

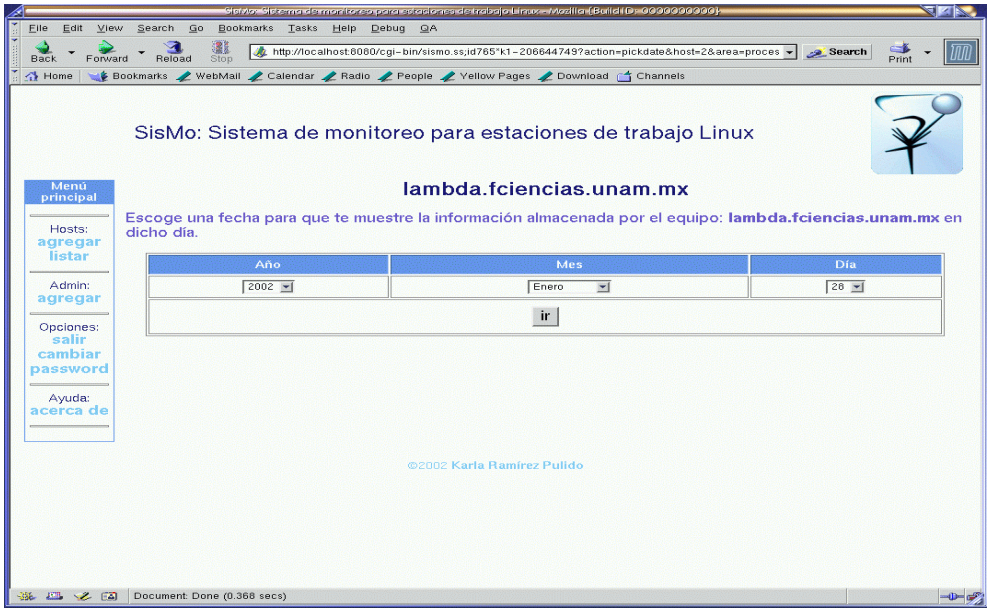

Figura 10.6: Después de haber seleccionado procesador, SisMo le pide que escoja una fecha.

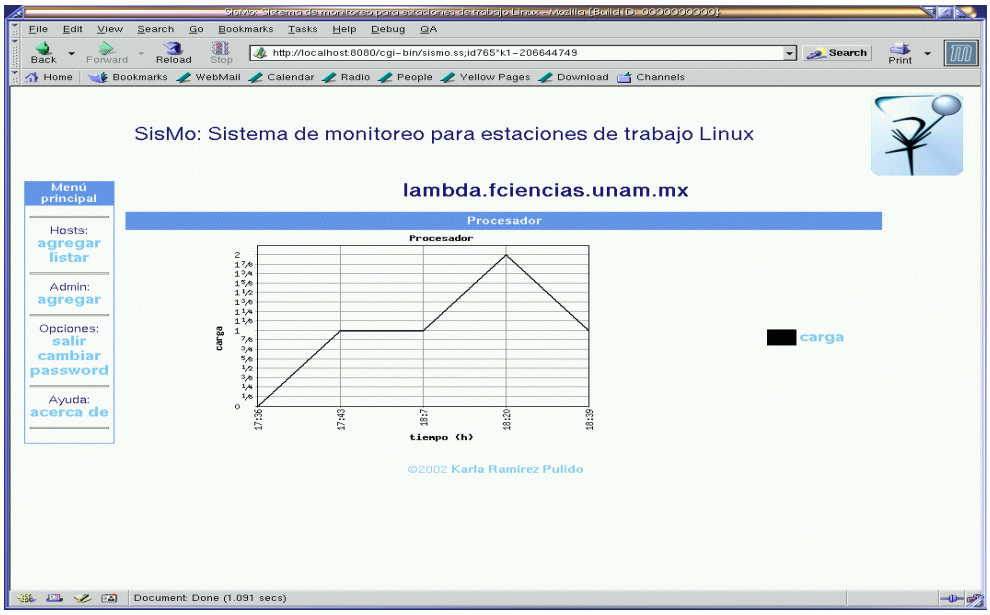

Figura 10.7: La información de procesador de lambda.fciencias.unam.mx almacenada para la fecha seleccionada.

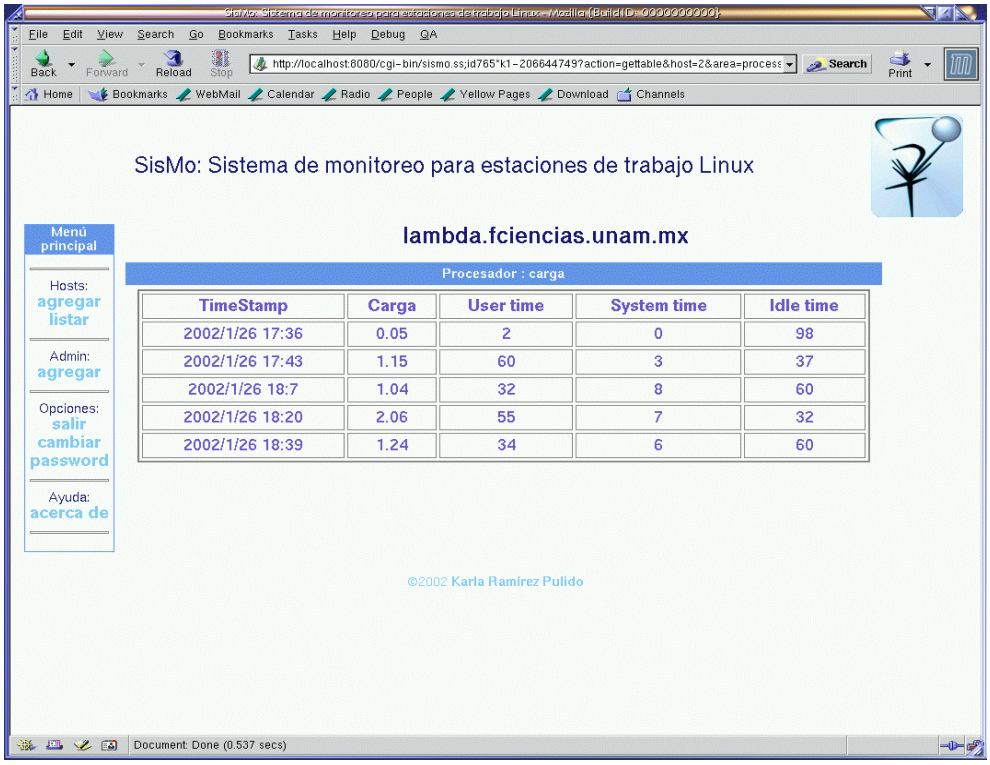

Figura 10.8: Detalles del procesador usados para graficar. Para efectos del ejemplo usamos datos inventados, en la práctica, esos datos serán enviados por el equipo a intervalos regulares y abarcarán las 24 horas del día seleccionado.

10.3.2 Cliente (de Web) del SisMo

El cliente de Web para el SisMo, es sin lugar a dudas, el cliente más sencillo de todo el sistema. Se muestra a continuación:

```
(define sismo-client
  (lambda (stuff)
    (let-values
     ([(ip op)
       (tcp-connect (sismo-server-name) (sismo-port))])
     (dynamic-wind void
                   (lambda ()
                     ;; Enviando información al demonio:
                     (display stuff op)
                     (newline op)
                     ;; Leyendo la respuesta:
                     (read ip))
                   (lambda ()
                     (close-input-port ip)
                     (close-output-port op))))))
```

Como se puede apreciar, lo único que hace es conectarse al puerto donde escucha el servidor (en el servidor donde está instalado el demonio), envía la información (previamente arreglada para ordenar alguna acción al SisMo) y luego lee la respuesta (que siempre es una única expresión).

10.3.3 El CGI

El CGI principal del SisMo y único CGI del sistema funciona (siempre) así:

1. Recibe una solicitud de página.
2. Analiza los parámetros pasados al CGI (soporta tanto interfaz GET, como POST).
3. Genera una expresión-X, que son expresiones soportadas por la biblioteca para soporte de *XML* que viene con PLT Scheme. Este mismo tipo de expresiones son esperadas como resultado de la ejecución de un CGI por el servidor de Web.

Todas las páginas en el sistema se generan con la siguiente expresión (por eso es tan uniforme la salida):

```
`(html
  (head
   (meta ((name "author") (content "Karla Ramírez (SisMo, powered by PLT Scheme)")))
   (meta ((name "keywords") (content "SisMo, sistema de monitoreo")))
   (link ((href "/style.css") (type "text/css") (rel "stylesheet")))
   (title "SisMo: Sistema de Monitoreo para Linux"))
  (body ((bgcolor "#ffffff"))
```

```
;; Encabezado
,@title
;; Cuerpo
(table ((width "100%") (border "0") (cellpadding "5"))
       (tbody ()
              (tr ()
                  (td ((valign "top") (width "110"))
                      ,@menu)
                  (td ((valign "top") (width "100%"))
                      ,@real-body))
              ,@footer))))
```

por supuesto, que `title`, `menu`, `real-body` y `footer` son variables que se generan con llamadas a otras funciones, algunas de las cuales incluso interactúan con el demonio del SisMo, o con el programa para generar las gráficas y que son a su vez expresiones-X, que contienen el verdadero contenido de la página.

10.3.4 Continuaciones en el CGI

En la sección anterior se muestra la finalidad del CGI del SisMo, sin embargo después de hacer *login* exitosamente, el resultado no es una página cualquiera, es una continuación que se parece mucho a todas las demás páginas del SisMo, pero son todas las ligas en esta página haciendo referencia no al único CGI (*sismo.ss*) sino a una continuación de la primera ejecución de ese CGI que está en memoria.

Por tal motivo, cuando se verifica que el *login* y el *password* de un administrador concuerdan con los que se tienen almacenados en una tabla de la base de datos, no hay más que iniciar una nueva continuación con un menú más complejo (que sólo puede ser generado por medio del CGI) y que tiene referencias a operaciones confidenciales y que conciernen ya a la parte escencial del sistema, que es la administración de los clientes (estaciones de trabajo Linux). Véase nuevamente que en la Figura 10.3, las ligas en el menú no apuntan al CGI original, sino a la continuación de la primera ejecución de dicho CGI (fue ejecutado para poder checar el nombre *login* y la contraseña *password*). Por eso es posible continuar autentificando al usuario: dado que ya proporcionó su *login*, su *password*, se verificó que fueran correctos (usando al cliente de Web mostrado en la Sección 10.3.2) y se tiene ya su identificador interno en una variable.

De hecho, a partir de hacer *login* exitosamente, y hasta que se escoja la opción *logout* en el menú, todas las páginas son generadas por llamadas anidadas (recursivamente) a la siguiente función:

```
;; generate-logged-in-page: Nat Xexpression -> continuation Esta
;; función recibe un número natural, el UID (user ID) de un usuario
;; que exitosamente logró entrar al Sismo. A partir de este momento
;; le atenderemos sólo con continuations como medida de seguridad, ya
;; que estos se generan dinámicamente. Para mayores detalles consulte
;; el capítulo 10 de la tesis: "SisMo: Un sistema de monitoreo para
;; estaciones Linux".
```

```
(define generate-logged-in-page
  (lambda (uid body)
    (let ( ;; Link es una pequeña trampa. Será una liga que siempre
          ;; apunte al continuation actual (i.e. uno menos que el
          ;; continuation al que se dirigen la mayoría de las ligas en
          ;; el menú, pero no es útil para poder controlar a todas las
          ;; demás ligas (dinámicas), por ejemplo, las que generan
          ;; llamadas a "sismo-show-host", etc.
          (link "/cgi-bin/sismo.ss"))
      (let-values
       ([(method url headers bindings)
         (send/suspend
          (lambda (k-url)
            (set! link k-url)
            ;; Media hora para que el administrador haga lo que tenga que hacer.
            (adjust-timeout 1800)
            `(html
              (head
               (meta ((name "author") (content "Karla Ramírez (SisMo, powered by PLT Scheme)")))
               (meta ((name "keywords") (content "SisMo, sistema de monitoreo")))
               (link ((href "/style.css") (type "text/css") (rel "stylesheet")))
               (title "SisMo: Sistema de Monitoreo para Linux"))
              (body ((bgcolor "#ffffff"))
                    ,@title
                    (table ((width "100%") (border "0") (cellpadding "5"))
                           (tbody ()
                                  (tr ()
                                      (td ((valign "top") (width "200"))
                                          ,@(generate-continuation-menu k-url))
                                      (td ((valign "top") (width "100%"))
                                          ,@body))
                                  ,@footer))))))])
       (let ((action (extract-binding 'action bindings))
             (real-body `((p (strong "Error:") " operación no soportada."))))
         (when action
               (cond ((string=? action "about")
                      (set! real-body (read-page "about")))
                     ((string=? action "help")
                      (set! real-body (read-page "help")))
                     ;; Lista de hosts administrados por el usuario con UID:
                     ((string=? action "list")
                      (set! real-body (sismo-get-hosts uid link)))
                     ;; Agregar algo (host o administrador);
                     ((string=? action "add")
```

```
                    (let ((what (extract-binding 'what bindings)))
                      (cond (;; Agregar host:
                             (string=? what "host")
                             (set! real-body (sismo-add-host-form link)))
                            (;; Agregar administrador:
                             else
                             (set! real-body (sismo-add-admin-form link))))))
                   ;; Agregando los datos de un nuevo host:
                   ((string=? action "addhost")
                    (let ((hostname (extract-binding 'hostname bindings))
                          (ip (extract-binding 'ip bindings)))
                      (set! real-body
                            (if (and hostname ip)
                                (store-host uid hostname ip)
                                `((h1 "Error")
                                  (p "Debes pasar tanto el hostname
como la dirección IP del host a monitorear.  Por favor regresa e
inténtalo de nuevo."))))))
                    ;; Agregando los datos de un nuevo administrador:
                    ((string=? action "addadmin")
                     (let ((login (extract-binding 'login bindings))
                           (name (extract-binding 'name bindings)))
                       (set! real-body
                             (if (and login name)
                                 (store-admin login name)
                                 `((h1 "Error")
                                   (p "Debes pasar tanto el login,
como el nombre del administrador que quieres agregar.  Por favor
regresa e inténtalo de nuevo."))))))
                    ;; Cambiar password
                    ((string=? action "changepass")
                     (let ((step (extract-binding 'step bindings)))
                       (when
                        step
                        (set! real-body
                              (cond ((string=?  step "form")
                                     (sismo-change-form link))
                                    (else
                                     (let ((old (extract-binding 'old bindings))
                                           (new (extract-binding 'new bindings))
                                           (confirm (extract-binding 'confirm bindings)))
                                       (if (and old new confirm)
                                           (if (string=? new confirm)
                                               (sismo-change-password uid old new)
```

```
                                                    `((h1 "Error")
                                                      (p "El nuevo
password NO concuerda.  Por favor, regresa y teclea con cuidado (ambas
veces) el nuevo password, gracias.")))
                                                        `((h1 "Error")
                                                          (p "Para poder cambiar
tu password, debes llenar todos los campos: tu password actual, y dos
veces el password nuevo.  Por favor, regresa y vuelve a
intentarlo.")))))))))))
                        ;; Para mostrar un host:
                        ((string=? action "show")
                         (let ((host (extract-binding 'host bindings)))
                           (when (and host
                                      (string? host))
                                 (set! real-body
                                       (sismo-show-host link (string->number host))))))
                        ;; Para seleccionar el día a desplegar la información:
                        ((string=? action "pickdate")
                         (let ((host (extract-binding 'host bindings))
                               (area (extract-binding 'area bindings)))
                           (when (and host area)
                                 (set! real-body (pick-date link host area)))))
                        ;; Para generar las gráficas:
                        ((string=? action "graph")
                         (let ((host (extract-binding 'host bindings))
                               (area (extract-binding 'area bindings))
                               (year (extract-binding 'year bindings))
                               (month (extract-binding 'month bindings))
                               (day (extract-binding 'day bindings))
                               )
                           (when (and host area)
                                 (let-values
                                  ([(from to) (make-date-from year month day)])
                                  (set! real-body
                                        (sismo-show-graph link host area from to))))))
                        ;; Para mostrar las tablas de datos:
                        ((string=? action "gettable")
                         (let ((host (extract-binding 'host bindings))
                               (area (extract-binding 'area bindings))
                               (label (extract-binding 'label bindings))
                               (from (extract-binding 'from bindings))
                               (to (extract-binding 'to bindings)))
                           (when (and host area label)
                                 (set! real-body
```

```
                                    (sismo-show-table link host area label from to))))))
    ;; Si action es "logout", entonces ya no enviamos un
    ;; nuevo continuation, simplemente regresamos a la
    ;; función que nos llamó (otro continuation o
    ;; 'sismo-login', la primera función que generó el
    ;; continuation.
    (unless (string=? action "logout")
            (generate-logged-in-page uid real-body)))))))
```

10.3.5 Gráficas

Un punto que no se ha tocado es la generación de las gráficas de los datos enviados por los clientes al SisMo. Por un lado, la graficación de los datos por sí misma nunca fué el problema más importante, sin embargo para efectos prácticos el soporte visual es lo más importante para los administradores que usen el SisMo. Los siguientes requerimientos fueron tomados en cuenta para decidir cómo generar dichas gráficas:

- La interfaz entre el SisMo y los administradores presenta una interacción dinámica, por lo que el tiempo de respuesta debe ser pequeño. Extraer los datos de las bases toma un cierto tiempo, graficar dichos datos debe ser una tarea muy rápida.
- Dado que durante la interacción se pueden generar muchas gráficas, también es muy importante que el consumo de recursos sea muy pequeño (memoria, tiempo de procesador, etc.)

De esta manera, varios programas que permiten la generación de gráficas simples a partir de datos fueron descartados, porque son sistemas muy complejos y que consumen más recursos de los necesarios, en particular varios sistemas de matemáticas. Por tanto, se optó por escribir un programa que hiciera justo lo necesario y no más.

El programa está escrito en C y utiliza la biblioteca *GDChart*[3], que a su vez usa *GD*[4]. GDChart es muy rápida y permite la generación de gráficas en formatos varios: GIF, PNG, JPEG y WBMP, por lo que es ideal para nuestros propósitos.

El programa es muy sencillo, por lo que sólo se presenta aquí una de las funciones en él `print_generic`; todas las demás funciones hacen llamadas a ésta, cambiando únicamente los parámetros que recibe:

```
void print_generic(char *infile, char *outfile, char *prog,
                   char *title, char *ytitle, char* xtitle) {
  FILE *outgif;
  FILE *rawdata;
  /* Estos arreglos contendrán la información que vamos a
   * graficar: */
  int data_points, filesystems, i, j, foo;
```

[3]GDChart fue escrita por *Bruce Verderaime* y es Free Software. Puede obtener mayor información en: `http://www.fred.net/brv/chart/`

[4]GD es una biblioteca de gráficos escrita por *Thomas Boutell* y es Free Software. Puede obtener mayor información en: `http://www.boutell.com/gd/`

```
float **data, **datum, *tmp;
char **lab, **labels, **dataTemp;
char *sep = "\t ";
char *p, *line;
unsigned long *sc;

if((outgif = fopen( outfile, "w")) == NULL) {
  fprintf(stderr, "%s: no puedo abrir el archivo: %s\n", prog, outfile);
  exit(65);
}
if((rawdata = fopen( infile, "r")) == NULL) {
  fprintf(stderr, "%s: no puedo leer del archivo: %s\n", prog, infile);
}

/* Ahora tenemos que leer los datos e ir guardando la
 * información en los arreglos adecuados.  Esta es la parte
 * truculenta y depende de que el archivo de datos tenga un
 * formato específico muy preciso.  No hacemos chequeo de
 * errores, porque sería muy lento. */
fscanf(rawdata, "%d\t%d\n", &data_points, &filesystems);

line = (char *)malloc(MAX_LINE_SIZE);
/* Leemos las eqtiquetas (de tiempo) del eje X */
if ((fgets(line, MAX_LINE_SIZE, rawdata)) == NULL) {
  fprintf(stderr, "%s: Error en formato del archivo de datos (etiquetas)\n", prog);
  exit(65);
}

/* Partimos en tokens la cadena de las etiquetas.  El
   formato que dee tener esta línea es cada etiqueta
   (timestamp) separada por TAB */
labels = (char **)malloc(sizeof(char **) * data_points + 1);
for(lab = labels; (*lab = strsep(&line, sep)) != NULL;)
  if (**lab != '\0')
    if (++lab >= &labels[data_points])
      break;
/* Liberamos el arreglo temporal */
free(line);

/* Ya conocemos el número de puntos y el número de sistemas de
 * archivos.  Le asignamos espacio a los arreglos de datos: */
data = (float **)malloc(sizeof(float **) * filesystems + 1);
/* Algo similar con sus colores */
sc = (unsigned long*)malloc(sizeof(unsigned long) * filesystems + 1);

for (i = 0; i < filesystems; i++){
  /* Nos comemos la línea */
  line = (char *)malloc(MAX_LINE_SIZE);
  if((fgets(line, MAX_LINE_SIZE, rawdata)) == NULL) {
    fprintf(stderr, "%s: Error en formato del archivo de datos (datos)\n", prog);
```

```
      exit(65);
    }

    /* Ahora nos comemos los datos.  Primero los metemos a un
       arreglo temporal y luego dividimos la etiqueta y los
       datos en los arreglos correspondientes para el GDCHART. */
    dataTemp = (char **)malloc(sizeof(char **) * data_points + 1);
    for(lab = dataTemp; (*lab = strsep(&line, sep)) != NULL;)
      if (**lab != '\0')
        if (++lab >= &dataTemp[data_points])
          break;

    /* Ahora nos comemos los datos de este sistema de
     * archivos: */
    data[i] = (float *)malloc(sizeof(float) * data_points);
    for(j = 0; j < data_points; j++) {
      foo = 0;
      sscanf(dataTemp[j], "%d", &foo);
      data[i][j] = (float)foo;
    }
    free(line); free(dataTemp);
    /* Le asignamos un color a esta línea */
    sc[i] = colors[i % MAX_COLORS];
  }

  /* Aplanamos los datos para poder imprimirlos */
  tmp = (float*)malloc(sizeof(float) * (filesystems * data_points) + 1);
  for (i = 0; i < filesystems; i++)
    for (j = 0; j <  data_points; j++)
      tmp[i*data_points + j] = data[i][j];

  /* Lo siguiente modifica algunos parámetros de gdchart para
   * hacer "adecuada" la salida. */
  GDC_title     = title;
  GDC_ytitle    = ytitle;
  GDC_xtitle    = xtitle;

  GDC_BGColor   = 0xFFFFFFL;  /* background (white) */
  GDC_LineColor = 0x000000L;  /* line color (black) */
  GDC_SetColor  = &(sc[0]);   /* assign set colors */


  /* Finalmente imprimimos la gráfica en el archivo de
   * salida. */
  GDC_out_graph( 500, 300,  /* short  ancho, altura */
                 outgif,    /* FILE*  open FILE pointer */
                 GDC_LINE,     /* GDC_CHART_T tipo de gráfica
                                  (sólo usamos GDC_LINE, o GDC_3DLINE) */
                 data_points, /* int número de puntos por
                                 conjunto de datos. */
```

```
                labels, /* char*[] arreglo de etiquetas (eje X) */
                filesystems, /* int número de conjuntos de datos */
                tmp, NULL ); /* float[] conjunto de datos
                                (aplanados a una sóla dimensión) */
}
```

La salida de esta función, una gráfica, se guarda en el archivo `outfile`. Tanto el archivo de entrada (que contiene los datos), como el de salida, son conocidos por el CGI que atiende al administrador, por lo que el CGI, puede regresar al administrador una página con una referencia a la figura generada por esta función, ya que su nombre y posición en el sistema de archivos son conocidos de antemano.

Y más aún, dado que la generación de las gráficas por la función arriba mencionada es tan rápida, para el administrador su creación es transparente: es más lenta la transmisión de datos por una red común (10/100Mb) que lo que tarda el programa en generar una gráfica promedio (y la mayoría aquí lo son).

Capítulo 11

Conclusiones

SisMo es un sistema que ayuda a mantener una visión uniforme e integral de todos los factores de riesgo en la administración de servidores Linux. Es uniforme, porque todos los rubros monitoreados por SisMo se presentan al usuario de manera similar. Es integral, porque todo es accesible desde un mismo punto y con una interfaz amigable y muy simple. Los factores de riesgo en un sistema Linux están relacionados, y no por casualidad, con las actividades y los personajes que conviven en un sistema Linux usual:

Usuarios: La parte medular de un sistema Linux en particular y un sistema Unix en lo general. Son en su mayoría los usuarios quienes ordenan la ejecución de los procesos. Todo proceso, todo archivo y en general toda actividad que se realiza en el sistema en cualquier momento le *pertenece* a (o se realiza a nombre de) un usuario.

Procesos: Cada usuario en el sistema puede ejecutar varios procesos que consumen recursos compartidos: memoria, tiempo de procesador, etc. Es deseable entonces que el administrador sea capaz de detectar fácil y oportunamente procesos que consuman algún recurso en exceso. Es difícil definir *exceso* en este sentido, porque hay procesos que demandan más recursos que otros y tampoco es fácil hacer una clasificación de procesos. Sin embargo, el SisMo le permite al administrador *conocer* el tipo de procesos que se ejecutan en el sistema, o al menos, su comportamiento, de tal suerte que un pico muy alto en la gráfica de procesos, en un momento dado, sólo puede significar una cosa: que un proceso (o conjunto de procesos) están tomando más recursos de lo acostumbrado y es buena práctica de administración investigar qué sucede, qué proceso es, para qué sirve, quién lo ejecuta, etc.

Los procesos son la huella de los usuarios en el sistema y están íntimamente relacionados con el resto los rubros en esta lista.

Procesador: El procesador se encarga de ejecutar diferentes procesos de manera eficiente. Sin embargo, para aprovecharlo es necesario monitorear su carga de trabajo, su utilización de memoria, etc. Es muy difícil sacar información provechosa directamente de este apartado por sí mismo, pero le permite al administrador detectar momentos conflictivos que le inviten a revisar los demás rubros que juegan un papel importante en el consumo de tiempo de procesador: procesos, usuarios, memoria, etc..

Sistema de archivos: En todo sistema operativo es muy importante asignar de manera eficiente el espacio en disco, pues es a través de éste que se puede almacenar cualquier tipo de información. Sin embargo el espacio en disco es un recurso finito y por consiguiente puede terminarse. Una mala planeación de las particiones en un sistema Linux, o un proceso escribiendo sin control pueden acabar por dejar en un estado de inutilización al sistema completo. Es por tanto de vital importancia mantener un monitoreo del sistema de archivos y asegurarse que ninguna de las particiones se sature, no al menos sin que el administrador esté enterado y pueda elaborar un plan de contingencia (agregar un disco duro extra, montar un sistema de archivos de otro sistema –por medio de NFS–, etc.).

Memoria: La memoria principal del sistema es quizá el recurso más complejo de monitorear, ya que su utilización puede variar dramáticamente de un instante a otro. Está, por supuesto, relacionada directamente con los procesos y no sólo por el número de procesos en ejecución, sino también por el tamaño de la información que éstos manejen. Un administrador experimentado usualmente comienza analizando la memoria, detectando procesos que estén consumiendo una cantidad considerable de ella y analizando por qué lo hacen. Un programa con errores de programación puede fácilmente consumir una cantidad desproporcionada de memoria en pocos instántes, y la única solución razonable es matar el proceso.

Red: Toda computadora conectada a través de una red tiene la propiedad de enviar y recibir paquetes que llevan algún tipo de información. Muchos de estos paquetes pueden perderse o llegar a su destino con la información alterada, o tardíamente, así como provocar colisiones en la máquina receptora; por tal motivo resulta indispensable monitorear este rubro. Para el administrador inexperto, puede parecer sombrío el hecho de ver la cantidad de paquetes que se pierden o colisionan; sin embargo, si esos números aumentan considerablemente en un periodo corto de tiempo, eso puede indicar un problema serio, ya sea en la red física, en la puerta de enlace (*gateway*) de la subred donde el equipo está conectado, o en el equipo mismo. En estos casos es imperativo conducir una investigación detallada para poder corregir el problema.

Sistema de archvivos en red (NFS): Permite montar sistemas de archivos remotos en el equipo local y exportar sistemas de archivos locales para otros equipos. NFS se ha consagrado como una herramienta excelente, sobre todo en sistemas grandes; por ejemplo en redes universitarias para centralizar los archivos de los estudiantes, el correo de toda la organización, etc. Tiene serios problemas de seguridad que escapan el alcance de este trabajo y que no son cubiertos por el SisMo. Sin embargo es muy usado, así que es fundamental mantener un estricto control sobre qué se exporta y qué está siendo importado (o montado) desde equipos remotos.

Impresoras: La tarea de administrar las impresoras es también parte del trabajo del administrador, pues requiere de cierto control sobre el comportamiento de las mismas, en particular del demonio de impresión quien controla la cola de impresión, porque es por medio de éste que el administrador puede tomar ciertas decisiones respecto al estado de esta.

La información obtenida de estos rubros es pasada al servidor a través de la red (usando el protocolo TCP/IP); una vez que el servidor la recibe se encarga de almacenarla en una base de datos, la cual es manipulada durante la obtención de datos para poder mostrar dicha información de manera gráfica al administrador. Por otra parte el administrador en el servidor tendrá que haber empezado una sesión en el SisMo a través de cualquier browser.

La cantidad de información que requiere un administrador para poder mantener en condiciones aceptables algún sistema a su cargo es considerable. El SisMo contribuye de manera eficiente a sintetizar y mostrar la información necesaria, y el nivel de detalle proporcionado es suficiente en el 90% de los casos para que el administrador pueda analizar el estado de cada una de las máquinas monitoreadas (clientes). El SisMo tiene la capacidad de prevenir al administrador de posibles situaciones anómalas durante el funcionamiento de cada uno de los clientes. El desempeño que tiene el SisMo es aceptable, pues cada uno de los programas implantados tienen un bajo impacto en las máquinas monitoreadas, dado que son realmente muy eficientes y pequeños. Por otra parte el servidor o monitor, además de ser robusto y de fácil uso para el administrador debido a la interfaz dispuesta para tales fines, cuenta con módulos muy eficientes que permiten una respuesta rápida y certera en la información monitoreada. De esta manera el SisMo cumple con los objetivos esperados pues muestra al administrador una visión global instantánea de los clientes a su cargo.

El SisMo es de mucha utilidad para la prevención de *desastres*, ya que por su característica de brindar una visión global instantánea de los clientes, permite exponer tal situación al administrador desde su gestación, ofreciéndole la posibilidad de tomar las acciones preventivas que considere pertinentes y brindándole la posibilidad de dar seguimiento al problema.

Los objetivos iniciales (vea las secciones 1.3 y 1.4) buscados con el SisMo son fácilmente verificables, ya que las cosas que se deben monitorear están todas ahí, en la interfaz que maneja el administrador. Sin embargo, algunos detalles pueden ser perfeccionados y se mencionan a continuación:

Seguridad: En la actualidad es imposible no considerar la seguridad como una de las principales, actividades en la administración de un sistema Linux. Sin embargo, es un tema tan complejo en sí mismo, que se tendría que escribir un trabajo completo para intentar analizarla. Por eso quedaron fuera del SisMo todos los problemas relacionados con seguridad usuales en un sistema Linux.

Puntos de monitoreo: Por razones prácticas, los puntos de monitoreo (tiempos en que se ejecuta un cliente dado en un equipo siendo monitoreado) tienen que ser relativamente espaciados y enviados al servidor del SisMo para su análisis posterior. El problema práctico es la presentación de los resultados, ya que si el número de puntos (en la gráfica equivalentes al eje de las X) es muy grande, o denso, los resultados son difíciles de leer.

Un sistema de monitoreo debería estar acompañado de un sistema de reacción, que permitiera a los administradores experimentados configurar el monitor para que, dadas ciertas condiciones *dispara* un proceso para aliviar un problema en el momento que éste se presenta. Por ejemplo, supóngase que un proceso comienza a escribir datos al disco a una velocidad muy alta; el monitor debería tener la capacidad de detectar

(rápidamente) la situación y después de cruzar cierto umbral (por ejemplo el disco al 90% de capacidad) disparar un proceso que "hiciera algo" para aliviar el problema, como por ejemplo matar al proceso que está escribiendo sin control. Nuevamente, un sistema de reacción así escapa, los alcances de este trabajo, pero no por eso deja de ser deseable.

Configurabilidad: El SisMo tiene un procedimiento de instalación rústico, pero suficiente para nuestros fines. En sistemas de producción debería ser más fácil de instalar, configurar y venir acompañado de programas para facilitar su instalación y configuración. En este mismo renglón, es importante comentar que algunas de las herramientas usadas por SisMo serán pronto descontinuadas, por lo que el SisMo deberá portarse a nuevas versiones una vez que sean estables. En particular, PLT Scheme, del que se usó la versión 103, misma que ya no será mantenida una vez que se libere la versión 200 (ya en versiones *beta*); al igual que SchemeQL y el servidor de Web que vienen con la versión 103.

Una cuestión más que sería deseable,en cuanto a la generación de gráficas, sería tener una interfaz para generar gráficas desde Scheme, porque lo que se gana al tener todos los datos en memoria (vea la Sección 10.3.4) con continuaciones de Scheme, se pierde al tener que enviar los datos a un archivo y luego llamar al programa que grafica los datos. El ideal sería: obtener los datos del servidor del SisMo (como se hace ahora), darle formato a esos datos y pasarlos a un procedimiento (en Scheme también) que se encargara de graficar los datos y el resultado enviarlo (sin pasar nunca por el disco) directo al puerto donde el cliente (browser del administrador) está esperando.

Dejando atrás los detalles que pueden mejorarse, el SisMo cuenta con lo siguiente: Su interfaz para administradores es amigable y muy sencilla, por lo que puede ser usada tanto por administradores expertos e inexpertos, que fue una de las metas iniciales. El uso de las continuaciones en el servidor de Web también permite gran flexibilidad (y facilidad) de programación, lo que hace más eficientes todos los procesos y redunda en un mejor tiempo de respuesta en modo interactivo del SisMo.

La arquitectura del SisMo lo hace robusto y le permite atender simultáneamente a muchos clientes, aún cuando algunos de éstos se mueran o sean interrumpidos a la mitad del envío de sus datos. Sin embargo, el SisMo espera que el número de datos enviados por algunos de sus clientes sea estable. Por ejemplo, espera que el número de sistemas de archivo (enviados por el cliente `fs`) sea el mismo, al menos durante el periodo que está siendo revisado por el administrador. Si los datos, que se esperan estables, no lo son, dichos problemas se verán en las gráficas, pero es más costoso asegurarse que los datos son correctos, antes de generar la gráfica. De la misma forma, se optó por no usar transacciones (de la base de datos) para recibir la información de los clientes, porque eso reduciría la concurrencia de clientes conectados al SisMo.

Apéndice A

Código fuente

Todo el código del SisMo (definición de las tablas para la base de datos, clientes y el demonio) puede encontrarse en el CD-ROM que se entrega junto con este trabajo. Sin embargo, aquí se explican algunas de las funciones más importantes del servidor del SisMo.

El servidor del SisMo es un programa que consta de varias secciones, sencillas todas ellas, pero muy importantes. Se encuentran todas en un solo archivo, pero para que sea más fácil su comprensión aquí las explicaremos separadas.

A.1 Demonio

Cuando el programa servidor del SisMo es iniciado (más adelante veremos cómo, en la Sección A.3, en el siguiente apéndice), un grupo de funciones se encarga de arrancar un nuevo hilo de control para atender a cada cliente que se conecta al puerto del SisMo.

La función `sismo-serve` es la que arranca el demonio del SisMo y básicamente lo que hace es crear un par de puertos para escuchar y leer en el puerto seleccionado, y luego lanza un nuevo hilo de control junto con una función, `server-loop`, que lee y escribe en estos puertos y atiende al primer cliente en la cola (los *sockets* en TCP tienen una cola de clientes en espera).

```
(define sismo-serve
  (opt-lambda
   ([port (sismo-port)]
    [max-waiting (sismo-max-waiting-clients)])
    (let ([custodian (make-custodian)])
      (parameterize ([current-custodian custodian])
                    (let ([listener (tcp-listen port max-waiting)])
                      (thread
                       (lambda ()
                         (server-loop listener)))))
      (lambda () (custodian-shutdown-all custodian)))))
```

La función `server-loop` acepta al primer cliente en la cola y crea un `custodian`, que es simplemente un proceso que maneja todos los puertos, descriptores de archivo, etc.,

para atenderlo. La intención de esto es que si el cliente "se cae", o algo sucede con la conexión, esto no afecte a la máquina virtual completa. Es decir, el servidor seguirá escuchando aún cuando una conexión particular se muera repentina e inesperadamente. Cabe recordar que, por cuestiones prácticas, el protocolo utilizado no tiene grandes medidas de seguridad, por lo que es fácil enviar información que confunda al SisMo: la conexión podría caerse o quedarse colgada mucho tiempo y por tal motivo se creará un nuevo custodian, que es eliminado en cuanto la conexión termina (por cualquier motivo), antes de irse, libera todos los recursos abiertos, incluidos los puertos de red.

```
(define (server-loop listener)
  (let-values ([(ip op) (tcp-accept listener)])
              (let ([cust (make-custodian)])
                (parameterize ([current-custodian cust]
                               [exit-handler (lambda (x) (custodian-shutdown-all cust))])
                              (thread (lambda ()
                                        (sismo-serve-connection ip op)
                                        (custodian-shutdown-all cust))))))
  (server-loop listener))
```

Como puede verse, la función `server-loop` llama a la función `sismo-serve-connection` para atender a cada cliente y vuelve a llamarse a sí misma (para seguir atendiendo indefinidamente), como un demonio de Unix.

A.2 Implementación del protocolo

La función `sismo-serve-connection` es la única función que implementa el protocolo que hemos discutido en este trabajo (vea las Secciones 10.2 y 10.2.1). Es una función muy grande, porque tiene un selector que soporta todas las posibles acciones que le pueden solicitar los clientes al SisMo. Su "forma" es la siguiente:

```
(define sismo-serve-connection
 (lambda (ip op)
   (dynamic-wind
    void

     (lambda ()
       (let loop ((ln (read-line ip)))
         (let* ((lst (string-tokenizer #\: ln))
                (command (car lst)))
           (cond (
                  ;; Dependiendo del valor de 'command', en este
                  ;; selector se analiza qué está pidiendo un cliente
                  ;; y lo llevamos a cabo, o marcamos un error.
                  )))))
     (lambda ()
       (close-input-port ip)
```

(*close-outputport op*)))))

El `dynamic-wind`, es una función estándar de Scheme que consta de tres partes (tres funciones que no reciben parámetros) y asegura que siempre se ejecuten en ese orden (uno, dos y tres), no importando qué hagan las funciones anteriores. Es una construcción muy útil en este caso, porque si algo sale mal durante la ejecución de la segunda función (que es la que hace el verdadero trabajo), y lanza una excepción o falla de alguna forma, entonces de las 30 conexiones que acepta el SisMo, por omisión perderíamos una, ya que los puertos para escuchar al cliente actual quedarían ocupados indefinidamente. Pero eso nunca sucederá, ya que la tercera función siempre se ejecuta y justamente cierra esos puertos.

Hay muchas funciones particulares que sirven para implementar el protocolo, pero ninguna requiere explicaciones particulares, aparte de los comentarios que ya tienen en el código fuente.

A.3 Servidores de Web y del SisMo

Un último detalle que debemos cubrir es la forma de iniciar el servidor de Web y el servidor del SisMo, lo cual dada la infraestructura con que contamos hasta este punto es un tarea muy sencilla y se logra con el siguiente guión de shell:

```
#!/bin/sh
":";exec mzscheme -r $0


;; Servidor de Web para el SisMo:

;; ********************
;; Biblioteca de Web
;; ********************
(require-library "web-server.ss" "server")

;; El servidor del Sismo:
(load-relative "daemon.ss")

;; La raíz del servidor de web está aquí:
(define sismo-virtual-system
  (lambda (host)
    "/home/karla/tesis/server/www"))

;; La función 'serve' regresa una función que
;; sirve para matar el servidor de Web.
(define sismo-web
  (serve 8080 30 sismo-virtual-system))

(printf "¡El SisMo Web Server está arriba!.~n")

(define kill-sismo (sismo-serve))
```

```
(let loop ((spin 0))
  (printf "~a> " spin)
  (let ((comm (read-line)))
    (cond ((string=? comm "exit")
           (sismo-web)  ;; Terminando el servidor de web
           (kill-sismo) ;; Terminando al servidor del sismo
           (exit 0))
          (else
           (printf "comando no reconocido~nPor el momento sólo entiendo: 'exit'.~n~n")))
    (loop (add1 spin))))
```

que como puede notarse, es un ciclo infinito que sólo reconoce el comando *exit* y cuando eso es tecleado en el prompt que ofrece, termina tanto al servidor de Web, como al servidor del SisMo. Mientras no se terminen tanto el servidor de Web como el del SisMo, se mantienen como demonios escuchando en los puertos 8080 y 8989, respectivamente.
La construcción:

```
#!/bin/sh
":";exec mzscheme -r $0
```

al inicio del archivo logra que el shell inicie la ejecución del script, pero la siguiente línea (la que comienza con `":"`) inmediatamente le pasa el control a *mzscheme* para que se haga cargo del resto.

Bibliografía

[1] Paul W. Abrahams and Bruce R. Larson. *Unix for the impatient*. Addison-Wesley, 2nd. edition, 1996.

[2] Maurice J. Bach. *The Design of the UNIX operating system*. Prentice-Hall Software. Prentice-Hall, Inc., 1986.

[3] Douglas E. Comer. *Internetworking with TCP/IP*, volume I of *Principles, Protocols, and Architecture*. Prentice Hall, 2nd. edition, 1991.

[4] David Fiedler and Bruce H. Hunter. *UNIX System V Release 4 Administration*. SAMS, 2nd edition, 1994.

[5] Lehey Greg. *The Complete FreeBSD*. Walnut Creek CDROM Books, 3rd. edition, 1999.

[6] Craig Hunt. *TCP/IP Network Administration*. O'Reilly & Associates, Inc., 1st, minor corrections edition, May 1994.

[7] Bill McCarty. *Learning Red Hat Linux*. O'Reilly & Associates, Inc., 1st. edition, September 1999.

[8] Evi Nemeth, Garth Snyder, Scott Seebass, and Trent R. Hein. *UNIX System Administration Handbook*. Prentice Hall PTR, 3rd. edition, 2001.

[9] Steve Van Der Hoeven Paul Graunke, Shriram Krishnamurthi and Matthias Felleisen. Programming the web with high-level programming languages. In *Proceedings of FP*, 2001.

[10] David A. Rusling. *The Linux Kernel*. on-line, 1999. http://www.linuxhq.com/guides/TLK/tlk.html.

[11] Gray John Shapley. *Interprocess comunications in UNIX. The nooks and crannies*. Prentice HALL, 1997.

[12] Charles Spurgeon. Networking services. http://www.uwsg.indiana.edu/usail/external/ethernet/.

[13] Andrew S. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, 1st. edition, 1998.

[14] Indiana University. Unix workstation system administration education certification course. IUB Webmaster Copyright 1996, The Trustees of Indiana University. http://www.uwsg.iu.edu/usail/edcert/, 1996.

Índice

R

S

T

U

V