Universidad Politécnica de Madrid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera

Creación de nuevas funcionalidades para la herramienta VNUML

Autor:
Esteban Martín Malo

Madrid, June 29, 2006

Universidad Politécnica de Madrid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera

Departamento de Ingeniería de Sistemas
Telemáticos

Creación de nuevas funcionalidades para la herramienta VNUML

Autor:
Esteban Martín Malo

Tutor:
David Fernández Cambronero

Madrid, June 29, 2006

Título:	**Creación de nuevas funcionalidades para la herramienta VNUML**
Autor:	**Esteban Martín Malo**
Tutor:	**David Fernández Cambronero**
Departamento:	**Ingeniería de sistemas telemáticos**

Miembros del tribunal

Presidente: Joaquín Seoane Pascual
Vocal: David Fernández Cambronero
Secretario: Juan Carlos Dueñas López
Suplente: Luis Bellido Triana
Fecha de Lectura:
Calificación:

A todos aquellos que han estado a mi lado en estos largos 6 años, en especial a "los nueve".
A María, y a toda mi familia
Esteban, María, Fernando y Pablo.
Y por supuesto, a los integrantes del proyecto VNUML, sin los que no hubiera sido posible la realización de este trabajo.

Indice

1 Introducción y objetivos **10**
1.1 Introducción 10
1.2 Estructura el documento 12

2 Estado del arte **14**
2.1 Introducción 14
2.2 Virtualización mediante hardware 21
2.3 Virtualización mediante software 25
2.3.1 Introducción 25
2.3.2 VMware 25
2.3.3 Xen 27
2.3.4 Virtual Iron 29
2.3.5 Microsoft Virtual PC y Virtual Server 30
2.3.6 Virtuozzo y OpenVZ 30
2.3.7 Otras alternativas 31
2.3.8 User Mode Linux 33
2.3.9 Virtual Network User Mode Linux 36
2.3.10 Casos de uso 37

3 CD Autoarrancable **40**
3.1 Introducción 40
3.2 Estudio de CDs autoarrancables 41
3.3 Implementación 45
3.4 Kernel y proceso de arranque del CD 50
3.5 Conclusiones 56

4 Analizador de protocolos **60**
4.1 Introducción 60
4.2 La librería pcap 63

4.3 Implementación 70
4.3.1 Captura de paquetes 70
4.3.2 Filtrado de paquetes 73
4.3.3 Captura de tráfico en tiempo real 73
4.3.4 Envío de mensajes a una instancia del uml_switch en ejecución . 76
4.3.5 Últimas modificaciones 85
4.4 Conclusiones 86

5 Interfaz Gráfica de usuario 89
5.1 Introducción 89
5.2 Las librerías GTK 90
5.3 Estudio de Vnumlgui 93
5.4 Modificaciones implementadas 98
5.5 Añadiendo nuevas funcionalidades : etiquetas de captura en uml_switch 100
5.6 Conclusiones 108

6 Conclusiones y Futuros Trabajos 110

A Remastering Knoppix Howto 114
A.1 Introduction 114
A.2 Setting up the environment 114
A.2.1 Installing and removing Debian packages 115
A.2.2 Setting user environment 117
A.3 Kernel remastering 118
A.3.1 Applying kernel patches 118
A.3.2 Modifying minirt.gz 122
A.4 Final CD-ROM Image 123

Indice de figuras

3.1 Imagen bienvenida CD 57
3.2 Imagen konqueror 58
3.3 Simulación en el CD 59

5.1 Interfaz Glade 102
5.2 Imagen Vumlgui 109

Capítulo 1

Introducción y objetivos

1.1 Introducción

Hoy en día la metodología de la virtualización está experimentando un progreso notable, no quedando relegada al entorno de los mainframes como pasaba antiguamente, sino que su uso se está extendiendo a muchos más ámbitos. Prueba de ello son los numerosos productos software dedicados a tal efecto, con aproximaciones muy dispares, y el hecho de que los mayores fabricantes de procesadores, Intel y AMD, han desarrollado productos orientados a facilitar el proceso de la virtualización.

El desarrollo completo de este proyecto se basa en una herramienta de libre distribución utilizada para la simulación de redes virtuales, llamada VNUML, Virtual Network User Mode Linux. El objetivo general es aumentar las posibilidades de utilización de la herramienta. Esto incluye desde facilitar su uso a cualquier usuario sin exigirle conocimientos específicos, hasta estudiar cómo mejorar la funcionalidad del proyecto en diversos aspectos.

Para ello, el proyecto seguirá un desarrollo gradual. En primer lugar, se analizará el estado del arte del mundo de la virtualización. Se estudiará cuales son los principios en los que se basa, y cuales son las alternativas existentes en forma de productos cuyo fin sea la virtualización. Se analizarán las ventajas e inconvenientes que acarrea esta metodología, indicando los casos de uso más habituales. Además, se realizará un estudio particular del método de simulación de User-Mode-Linux, herramienta que permite simular máquinas virtuales GNU/Linux, y que es la utilizada por VNUML para la creación de los equipos simulados. Y por supuesto, se analizará la propia herramienta, estudiando su comportamiento, sus posibilidades, y en qué puntos se podrían añadir nuevas funcionalidades.

Posteriormente, se analizará uno de los mayores inconvenientes que plantea el uso de la herramienta, como es su instalación. Al ser una herramienta con numerosas funcionalidades, para su óptimo uso son necesarios una serie de requisitos, como un sistema

operativo Linux, soporte para el lenguaje Perl con unos módulos determinados, el software encargado de simular las redes, y los ficheros necesarios para simular las máquinas virtuales. Además, es muy recomendable, aunque no estrictamente necesario, que el kernel del sistema operativo tenga ciertas modificaciones para que la gestión de memoria del equipo también sea óptima. Todo esto es muy complicado para un usuario novel, o que simplemente quiere probar la herramienta, y se considera fundamental facilitar este proceso. Con este fin, el uso de las distribuciones autoarrancables existentes en el entorno GNU/Linux se nos presentan como la óptima alternativa. Ello implicará un estudio de las diferentes distribuciones en el mercado, las posibilidades de remasterización, o cómo realizan sus procesos de arranque, por ejemplo. El objetivo último de este apartado es la creación de una imagen autoarrancable para CD, que permita utilizar la herramienta en cualquier ordenador sin necesidad de instalar previamente el sistema operativo Linux. Esta imagen consistirá en el sistema operativo con todos los programas necesarios para su ejecución. Además, uno de los puntos de estudio que se llevará a cabo será las distintas posibilidades de persistencia de datos, de modo que el trabajo del usuario pueda ser guardado para posteriores utilizaciones de la herramienta. Para facilitar aún más el uso del CD a los usuarios, se comprobará que la imagen puede utilizarse desde otras herramientas de virtualización, como puede ser VMware, de modo que el usuario que así lo desee pueda utilizarlo directamente desde su sistema operativo habitual.

Un somero estudio de VNUML revela que para la simulación de la comunicación entre los distintos interfaces de red de las máquinas virtuales hace uso de un programa denominado uml_switch que imita el comportamiento de un switch ethernet. Una funcionalidad muy interesante destinada en principio para fines académicos, pero que puede tener muchas más posibilidades sería la posibilidad de capturar el tráfico que tiene lugar en la red. Así pues, para poder estudiar el tráfico comentado, se realizará una modificación en el uml_switch para capturarlo y poder observarlo posteriormente. Se incluirán además las posibilidades de la captura de tráfico en tiempo real, para visualizar el intercambio de paquetes según se produce, y de la aplicación de filtros para seleccionar los paquetes que se desean capturar. Para añadir más facilidades de uso, se considera interesante contar con la posibilidad de modificar el comportamiento relacionado con la funcionalidad de captura de un uml_switch en ejecución, indicándole, por ejemplo, que empiece a capturar en un determinado momento. Con tal fin, se estudiarán las posibilidades existentes en sistemas GNU/Linux de mecanismos de comunicación entre procesos, y se desarrollará el software correspondiente utilizando el mecanismo más apropiado para este caso.

Se estudiará a continuación el proyecto de interfaz gráfica de usuario, vnumlgui. Esta herramienta se considera muy interesante para facilitar el uso de VNUML, pues de este

modo se abstrae al usuario de la creación de los ficheros XML utilizados para definir la simulación. Dado que la interfaz gráfica es muy completa, admitiendo la mayoría de las opciones contempladas por VNUML, pero que ésta aún no se ha integrado completamente con la última versión de la herramienta, es también objetivo de este proyecto finalizar la integración de las mismas, para poder utilizarla con la más reciente versión de VNUML. Esta tarea implicará, además del estudio del propio código del proyecto, familiarizarse con el sistema de eventos utilizado por las librerías GTK, que son las utilizadas por vnumlgui para desarrollar la interfaz.

Por último, como tarea menor, se actualizarán los ejemplos de VNUML existentes en la web a la última versión estable de la herramienta, de modo que puedan ser probados por los usuarios sin ninguna complicación y comprueben algunos de los casos de aplicación de la herramienta, como es la creación de un entorno simulado en el que se utiliza el protocolo OSPF para el enrutamiento.

En un aspecto más técnico, este proyecto me permitirá estudiar diferentes aspectos de los sistemas operativos, en especial cómo se implementan en distribuciones GNU/Linux el proceso de arranque con sus gestores correspondientes, la comunicación entre procesos o el desarrollo de interfaces gráficas.

1.2 Estructura el documento

Procedemos a continuación a explicar la estructura del documento. El presente apartado forma parte del primer capítulo, en el que se definen los objetivos del proyecto, y los pasos en los que se dividen las tareas a realizar.

En el capítulo siguiente se procede al estudio del estado del arte de la virtualización. Se estudiarán sus bases teóricas, y se mostrarán las bases de diferentes aproximaciones, tanto de productos hardware como de proyectos software. Como es lógico, habrá un apartado dedicado por su especial importancia al proyecto User Mode Linux y otro más centrando en la herramienta VNUML.

Posteriormente, se adentrará en uno de los aspectos concretos de VNUML, como es el encargado de la simulación de redes. Una vez comprendido su funcionamiento, se explicará cómo se ha añadido la posibilidad de capturar el tráfico de las mismas, permitiendo además elegir qué tráfico filtrar, y hacerlo en tiempo real. Tras estudiar las posibilidades ofrecidas por sistemas GNU/Linux para la comunicación entre procesos, se procederá a la explicación de cómo se ha implementado en este caso concreto el envío de órdenes a las instancias uml_switch en ejecución.

Como último apartado relativo a las tareas, se explicará el proceso de actualización de la interfaz gráfica, que incluye tanto las modificaciones realizadas, como una guía

indicadora de los pasos a seguir para añadir las nuevas características que permitan las futuras versiones de VNUML.

Se dedicará a continuación una sección a las conclusiones generales obtenidas tras el desarrollo del proyecto, y se enumerarán una serie de tareas que se consideran interesantes para futuros trabajos relacionados con el presente desarrollo.

Esta memoria finaliza con un apéndice en el que se procede a la enumeración concreta de los pasos a seguir para la creación de un CD basado en la distribución Knoppix que incluya todo el software necesario para el uso óptimo de la herramienta.

Capítulo 2

Estado del arte

2.1 Introducción

La virtualización es una metodología que debido a las numerosas ventajas que ofrece, es cada vez más utilizada en diversos entornos de producción. Prueba palpable de ello es la cada vez más amplia variedad de tecnologías que ofrecen en diversos grados la posibilidad de crear máquinas virtuales, tanto a nivel de software, como VMware o Xen, o a nivel de hardware, con las tecnologías Pacifica o VT (Virtualization Technology). En esta sección se hará una introducción al estado actual de esta metodología, intentando explicar las ventajas e inconvenientes de cada una de las posibilidades más importantes existentes en el mercado.

Una posible definición de virtualización sería "un framework o metodología de dividir los recursos de un ordenador entre múltiples entornos de ejecución, aplicando uno o más conceptos de tecnologías como particiones software o hardware, compartición de tiempo, simulación parcial o completa de una máquina, emulación, calidad de servicio, y numerosas otras" [18]. Actualmente se desaprovecha alrededor del 70% de los recursos de un ordenador.

Hay que tener en cuenta, a pesar de la definición, que la virtualización no siempre implica el particionado. Por ejemplo, la computación Grid habilita la virtualización de sistemas distribuidos.

Entre las ventajas que ofrece la virtualización, se encuentran las siguientes:

- Las máquinas virtuales se pueden usar para consolidar la carga de trabajo de varios servidores que no utilicen al máximo los recursos de sus respectivos equipos. Los beneficios relacionados con este aspecto (percibidos o reales, pero a menudo citados por los distribuidores) son el ahorro de hardware, costes de entorno, de gestión y de administración de la infraestructura de servidores.

- La necesidad de ejecutar aplicaciones antiguas es otra posibilidad ofrecida por las máquinas virtuales. Una aplicación antigua no siempre se ejecuta en las últimas novedades de hardware o sistemas operativos. Incluso si lo hace, puede infrautilizar el servidor, por lo que tendría sentido utilizar varias aplicaciones. Esto podría ser difícil sin virtualización, pues tales aplicaciones no están programadas para coexistir en un entorno simple de ejecución.

- Las máquinas virtuales pueden ser usadas para proporcionar un entorno seguro y aislado para ejecutar aplicaciones no confiables. Se podría incluso crear un entorno dinámico de ejecución. Es un concepto importante, pues, en la creación de plataformas seguras de computación.

- Pueden ser utilizadas para crear sistemas operativos o entornos con recursos limitados, y con garantías de repartición de recursos. El particionado normalmente se utiliza conjuntamente con la calidad de servicio para la creación de sistemas operativos para dotarle de esa capacidad.

- Pueden ser utilizadas para proporcionar hardware simulado, o configuración de hardware con el que no se cuenta (como dispositivos SCSI, múltiples procesadores...) La virtualización puede ser usada para simular ordenadores independientes o redes enteras.

- Pueden ser usadas para ejecutar múltiples sistemas operativos simultáneamente, desde diferentes versiones de la misma familia de sistemas, hasta sistemas completamente diferentes.

- Son una poderosa herramienta para obtener información de debug y de la respuesta de las aplicaciones. Se pueden poner tales herramientas en el monitor de la máquina virtual y de este modo estudiar sistemas operativos en modo debug sin pérdida de productividad, o establecer escenarios más complicados.

- Pueden aislar los procesos a ejecutar, con la consiguiente limitación de la propagación del error y fallo. Se pueden provocar errores para comprobar cuál es su respuesta.

- Facilitan las migraciones de software.

- Son una gran herramienta para entornos académicos, por la seguridad que proporcionan al trabajar sobre ellas.

La emulación es un concepto que se suele considerar como sinónimo de virtualización, pero existe una gran diferencia, y es que en la emulación se utiliza software para simular la respuesta de hardware incompatible. Un ejemplo muy claro es Virtual PC, que permite ejecutar aplicaciones para Windows en un sistema operativo Mac sobre la arquitectura PowerPC. Como dato de interés, comentar que la mayoría de los emuladores están compuestos de los siguientes módulos:

- Un emulador de la CPU, que interpreta las instrucciones en código máquina generadas por la aplicación y ejecuta las operaciones equivalentes en el código del procesador del equipo físico.
- Un subsistema de memoria que proporciona la equivalencia entre las casi siempre diferentes arquitecturas de memoria física y memoria lógica.
- Emuladores de dispositivos entrada/salida, que gestiona las interrupciones generadas por las aplicaciones.

Así pues, se pueden entender las máquinas virtuales como software que crea un entorno simulado entre el ordenador y el usuario final que permite ejecutar software. Existen tres conceptos relativos a las máquinas virtuales:

- El significado original de *máquina virtual* es uno de los múltiples entornos de ejecución que tienen lugar paralelamente en un sólo equipo físico. Esto proporciona a cada usuario la sensación de que está utilizando una máquina física, cuando en realidad esta usando una máquina privada aislada de las demás. El software que gestiona las máquinas virtuales se denomina *virtual machine monitor* o *hypervisor*.
- Otra definición posible es el software que aisla la aplicación que esta siendo ejecutada por el ordenador. Como existen diferentes versiones de la máquina virtual para varios tipos de arquitecturas y procesadores, estas aplicaciones pueden ser ejecutadas en distintas plataformas. La aplicación es ejecutada mediante un interprete o *Just In Time compilation*. El ejemplo más conocido es el de la máquina virtual de Java.
- Por último, el término máquina virtual puede referirse también al software usado para simular un sistema operativo completo para el usuario final, permitiendo ejecutar simultaneamente varios sistemas operativos en el mismo equipo. Esto puede llevarse a cabo de tres formas distintas:

- Virtualización completa. La máquina virtual simula el hardware completamente, por lo que permite la ejecución de sistemas operativos sin necesidad de modificaciones, e incluso diseñados para CPUs de diferente arquitectura con diferentes instrucciones. En este caso es necesario la implementación de las funcionalidades propias de algún emulador.
- Paravirtualización.En este caso, la máquina virtual no simula el hardware, sino que utiliza una API, que requiere de modificaciones del sistema operativo original.
- Virtualización nativa. La máquina virtual sólo simula el hardware necesario para ejecutar los sistemas operativos aislados, pero el sistema operativo simulado ha de estar diseñado para el mismo tipo de CPU.
- Virtualización a nivel de sistema operativo. En realidad es más un método de particionado especial que una forma de virtualización. Permite el uso de varios sistemas pero con el mismo kernel y recursos físicos de la máquina.

El trabajo fundamental de cualquier sistema es gestionar el hardware que utiliza. Esto incluye repartir el tiempo de uso de CPU, gestionar el uso de memoria y gestionar los dispositivos físicos de entrada y salida como monitores, dispositivos de almacenamiento o adaptadores de red. Si el sistema es virtualizado, se ejecuta sobre un *VM monitor* (término originado por los mainframes de IBM de los años 70), y el sistema operativo se convierte en un invitado del hardware físico y no lo gestiona por si mismo. Mientras que el trabajo de CPU scheduling concierne algo a la virtualización, es la gestión de memoria y de los dispositivos de entrada/salida lo que más influye en el rendimiento.

La manera más sencilla de afrontar el problema de la virtualización de dispositivos de entrada/salida es crear un dispositivo Ethernet virtual o un dispositivo SCSI que será utilizado por el SO invitado. El VM monitor entonces ejecutará el driver real, y se establecerá una relación cliente/servidor entre el driver virtual correspondiente al sistema operativo invitado y el driver real. Esta es una idea sencilla, pero puede ser un factor limitante de la velocidad, especialmente si el driver funciona a velocidades de multigigabits. El funcionamiento, pues, consiste en que el driver real recibe la información, el sistema operativo determina a quién va dirigido, lo copia en el espacio de memoria del driver simulado, y es el SO invitado el encargado de transmitir la información a la zona de memoria de la aplicación correspondiente.

El problema es que todos estos procesos de copia afectan a la velocidad del rendimiento. Dispositivos como routers y switches son rápidos porque tiene muy limitadas el número de copias en memoria que realizan. Este es el camino que están siguiendo los sistemas

operativos de servidores. Idealmente, una vez que el driver copia la información en memoria, no es copiada nunca más. Para ello se usan punteros, y se modifican los permisos de los datos. Pero en un entorno de máquina virtual, incluso eso puede ser computacionalmente caro porque entre el monitor VM, el SO del equipo y la aplicación tienen qué decidir que se hará con la información y qué proceso posee realmente el espacio de memoria donde la información está situada.

La gestión de memoria puede ser incluso más importante. Modernos sistemas operativos han usado la gestión de memoria virtual durante años. La idea parte de que los SO presentan a la aplicación una imagen de memoria virtual que es mapeada hacia la memoria física. Con memoria virtual, una aplicación puede tener acceso a más memoria que la que el sistema realmente tiene. Desde que el mapeo de memoria es virtual, dispositivos de almacenamiento más lentos, como discos duros, pueden ser mapeados a la memoria física.

Para facilitar la gestión de memoria, esta es dividida en páginas. Es función del SO mantener las páginas más frecuentemente usadas en memoria real, mientras que el resto estarán en el disco duro (en la memoria swap o de intercambio). Para hacer esto eficientemente, el sistema operativo necesita la ayuda del hardware del sistema. La CPU mantiene una tabla de las páginas de memoria correspondientes al proceso en ejecución. Si ese proceso intenta acceder a una página situada fuera de la memoria física, la CPU se lo comunica al SO, y el SO recibe la página del disco, y la carga en memoria.

Con el objetivo de presentar un entorno VM al SO invitado, esta noción debe ser extendida más allá. Otra tabla de páginas de memoria puede ser creada para cada sistema invitado, y el monitor VM puede gestionar las recolocaciones de páginas a través de éste. De este modo, utilizando el VM monitor para gestionar el mapeo de memoria en vez de las múltiples copias de memoria a memoria de la información, se consigue mejorar notablemente la velocidad del servidor.

Esta situación puede ser incluso mas complicada para máquinas NUMA (Non-Uniform Memory Access). Con estos sistemas, el tiempo de acceso a memoria no es uniforme para todas las CPUs o memoria física. Los procesadores Opteron de AMD fueron diseñados para sistemas con varias CPUs o máquinas NUMA. Las CPUs gestionan una parte de la memoria del sistema y pueden acceder cada una a la memoria de la otra a través de un hypercanal de conexión. Por comparación, Intel actualmente emplea un único gestor de memoria fuera de la CPU a través del cual se intercomunican todas las CPUs. Esta gestión de memoria del chip se convierte a menudo en un cuello de botella.

El método de AMD presenta menos cuello de botella, pero hace que la operativa del SO sea más complicada, ya que tiene que asegurarse que las páginas más frecuentemente accedidas no están situadas en la memoria principal, sino en la memoria directamente

asignada a la CPU correspondiente.

Como el hypercanal es muy rápido y el diseño de AMD permite acceder a cualquier parte de la memoria en un solo acceso, las medidas de prestaciones en las aplicaciones de AMD son mejores. Sin embargo, cuando los sistemas NUMA crecen, los saltos necesarios aumentan, las interconexiones se vuelven más lentas, y la gestión de memoria se vuelve más crítica. En función del entorno previsto de virtualización, esto debería ser tenido en cuenta.

Como ya se enunció, existen diferentes técnicas de virtualización que proporcionan características similares pero que difieren en el grado de abstracción y en los métodos utilizados. Sin tener en cuenta las técnicas de emulación, correspondientes a la simulación de sistemas operativos sobre arquitecturas distintas, las alternativas más importantes, y en las que más profundizaremos a continuación, son las siguientes:

- Máquinas virtuales. Las máquinas virtuales emulan hardware real o ficticio, lo cual implica igualmente los recursos reales del equipo. Mediante esta aproximación es posible ejecutar un sistema operativo sin ninguna modificación porque el SO no es consciente de que su entorno no es real. El principal inconveniente es que es necesario privilegios adicionales para determinadas instrucciones de la CPU, y no pueden ser ejecutadas en espacio de usuario, necesitando de un Monitor de máquinas virtuales (VMM) que analice el código y lo ejecute de modo seguro. Es la técnica empleada por los productos Vmware y Microsoft Virtual Server. Facilita la simulación de sistemas operativos a costa de un rendimiento más bajo.

- Máquinas paravirtuales. Esta técnica también requiere de un VMM pero la mayoría de su trabajo se realiza en el código del sistema operativo del entorno simulado, que modifica su kernel para soportar el el VMM y evitar el uso innecesario de instrucciones privilegiadas, que requerirían el uso del *ring 0* de la arquitectura x86. Se basa en la idea de la virtualización utilizada por IBM. Proporciona la posibilidad de ejecutar múltiples sistemas operativos, pero han de ser portados previamente. Ejemplos muy claros son Xen y UML. Las prestaciones son mayores que en el caso anterior, pero los sistemas a simular han de ser portados (modificado el código del kernel para soportar el hypervisor).

- Otra técnica empleada es la virtualización del nivel del sistema operativo. La mayoría de las aplicaciones que se ejecutan en un servidor pueden fácilmente compartir la máquina con otras aplicaciones, si están aisladas y seguras. Los sistemas de virtualización de SOs han sido diseñados para proporcionar el aislamiento necesario para ejecutar múltiples aplicaciones o copias del mismo sistema operativo en

el mismo servidor, pero no diferentes sistemas operativos. OpenVZ, fork de código libre del proyecto Virtuozzo, y Linux Vserver, son ejemplos de esta virtualización. Proporcionan el mejor rendimiento y escalabilidad y son mucho más fáciles de administrar.

Profundizando en el concepto antes mencionando, un hypervisor es un software que permite la ejecución de múltiples sistemas operativos en un mismo equipo al mismo tiempo. El término proviene de supervisor, que es como se suele referir al kernel del sistema operativo. Los hypervisores fueron originalmente desarrollados a principios de 1970 cuando la reducción de costes esta forzando que múltiples ordenadores de diversos departamentos fueran ejecutados en una misma máquina, el *mainframe*. Ejecutando múltiples sistemas operativos simultaneamente, el hypervisor trajo una robustez y estabilidad; aunque uno de los sistemas fallase, los otros podrían continuar su trabajo sin ningún tipo de interrupción. El primer ordenador diseñado específicamente para una completa virtualización fue el IBM S/360 modelo 67. Incluía una tabla de traducciones de direcciones de memoria para permitir direcciones virtuales de memoria y otras técnicas que permitían una completa virtualización de las tareas del kernel, incluyendo la gestión de entrada/salida y el manejo de interrupciones. Hasta este momento, la virtualización sólo permitía ejecutar múltiples aplicaciones de nivel de usuario, a partir de entonces se consiguió la virtualización del modo supervisor o kernel.

Tras esta introducción, se procede a explicar dos aproximaciones diferentes a la virtualización, como son las recientes técnicas mediante hardware, y las más maduras mediante software. Dentro de éstas, se hará un apartado especial para la metodología User-Mode-Linux, por su especial importancia para el desarrollo de este proyecto. Por último, se dedicará una pequeña sección para enumerar diferentes casos de uso de estas tecnologías.

2.2 Virtualización mediante hardware

Reforzando la idea de la importancia de la virtualización, Intel y AMD están trabajando en hacer la virtualización más fácil en CPUs X86, y han diseñado microprocesadores especialmente orientados a este fin. Así pues, Intel ha utilizado la tecnología de virtualización (VT), anteriormente conocida como Vanderpool y Silverdale [19], mientras que AMD ha desarrollado la tecnología *Pacifica Secure Virtual Machine* (SVM) [21].

En función del nicho de mercado escogido por cada campaña de marketing, son tecnologías que pueden parecer diferentes. Intel está promocionando VT como una arquitectura segura para portátiles, mientras que AMD intenta situar Pacifica en el mercado de los servidores. En cualquier caso, ambas buscan facilitar la virtualización sobre sus equipos.

A continuación, se expone un breve resumen de los modos utilizados por la CPU y los anillos de protección, ya que sobre estos conceptos se basarán las mejoras propuestas por ambas tecnologías.

Para mejorar la estabilidad y seguridad de los sistemas operativos, éstos deben ser capaces de distinguir entre las operaciones llevadas a cabo por ellos mismos, o por los usuarios o aplicaciones. Esto puede ser difícil, pues entre las tareas propias del sistema operativo se encuentran acceder a segmentos de memoria, enviar instrucciones a la CPU o acceder a dispositivos de almacenamiento secundario, mientras que un usuario, mediante programas como navegadores web o procesadores de textos, pueden estar realizando el mismo tipo de actividades simultáneamente.

Los sistemas operativos tienen varios mecanismos de protección para asegurar que los procesos no afecten negativamente a los otros componentes del sistema. Uno de estos es por ejemplo la protección de memoria, y otro son los anillos de protección. Estos anillos imponen estrictos límites definiendo qué procesos pueden ejecutarse dentro de cada anillo, y qué instrucciones pueden invocarse correctamente. Los procesos ejecutados en los anillos más interiores tienen mayores privilegios que los que se ejecutan en los más exteriores, pues los anillos interiores sólo permiten que los componentes y procesos más fiables se ejecuten en su interior. Los sistemas operativos pueden variar en cuanto al número de anillos de protección, normalmente las procesos ejecutados en los anillos más interiores se les considera pertenecientes al modo privilegiado, de superusuario o del kernel, mientras que los correspondientes a los más exteriores corresponden al modo de usuario [2].

La arquitectura actual de anillos que es usado por un sistema es definida por el procesador y el sistema operativo. El chip hardware del procesador está construido para proporcionar un cierto número de anillos, y el sistema operativo debe ser desarrollado para

trabajar con esta arquitectura. Esta es una de las razones por las que un sistema puede funcionar sobre un chip Intel, y no sobre un chip Alfa, por ejemplo. Tienen diferentes arquitecturas y maneras de interpretar el conjunto de instrucciones.

Los componentes del sistema operativo operan en un anillo que les da acceso a posiciones de memoria, periféricos, drivers del sistema y parámetros de configuración especialmente sensibles. Como este anillo ofrece acceso a componentes muy críticos, es el más protegido. Las aplicaciones normalmente operan en el anillo 3, que limita el tipo de memoria, periféricos o drivers que pueden ser utilizados, y que es gestionado a través de componentes del sistema operativo o de llamadas al sistema. Si una aplicación intenta enviar instrucciones a la CPU que no corresponden a su nivel de permisos, la CPU trata esta violación como una excepción y puede mostrar un fallo protección general e intentar finalizar la aplicación.

Los anillos de protección más comúnmente usados por las diferentes arquitecturas son los siguientes:

- Ring 0 : Kernel del sistema operativo.
- Ring 1 : Restantes partes del sistema operativo.
- Ring 2 : Drivers de entrada y salida y diversas utilidades
- Ring 3 : Aplicaciones y actividad de usuario.

Los anillos fueron el concepto más revolucionario y visible introducido por el sistema operativo Multics, predecesor de la familia Unix. La mayoría de las arquitecturas de CPU incluyen algún tipo de protección de anillos, aunque los sistemas operativos no lo aprovechen completamente. Multics contaba con 8 anillos, pero la práctica totalidad de los sistemas hoy en día cuenta con bastantes menos. El kernel Linux actual tan sólo utiliza los anillos 0 y 3. El hardware es consciente del anillo asignado al proceso en ejecución gracias a los registros de la CPU, imponiendo además severas restricciones en el modo en que se puede pasar de un anillo a otro, y también en las direcciones de memoria a las que tienen acceso. Típicamente hay una instrucción de llamada al sistema que transfiere el control de un modo seguro cuando se intenta ejecutar instrucciones correspondientes a anillos internos.

Conociendo este método de protección, procedemos a continuación a explicar cómo las tecnologías comentadas utilizan estas funcionalidades. En un principio, intentar asociar la virtualización al hardware podría parecer contradictorio, ya que hasta ahora, todas las técnicas de virtualización han sido simular la capa de hardware mediante software. Pero no hay que olvidar que hay un problema en la simulación sobre arquitectura x86

porque el kernel del sistema operativo espera controlar directamente la CPU, o lo que es lo mismo, estar situado en el *anillo 0*, donde existe mayor funcionalidad. Un chip x86 tradicional no puede ejecutar un sistema operativo virtualizado en el ring 0 porque es necesario que en este ámbito se encuentre el hypervisor, que es el encargado de gestionar las máquinas virtuales. Por esto, parece que la alternativa idónea es ejecutar el sistema operativo invitado en alguno de los dos anillos vacantes intermedios.

Desafortunadamente, algunas instrucciones en código máquina de la arquitectura x86 sólo funcionan en el Ring 0. Para ejecutarse en anillos superiores, el SO debe ser reescrito, o al menos recompilado para evitar estas instrucciones, que es la aproximación seguida por la paravirtualización. Esto es muy popular en el mundo de Linux, pues IBM usa una técnica similar para ejecutar clusters Linux en los mainframes, pero requiere del conocimiento del código fuente de los sistemas operativos.

Para ejecutar un sistema operativo sin modificar fuera del ring 0, el hypervisor debe interceptar las instrucciones prohibidas y emularlas. Esta es la técnica utilizada por VMware o la utilizada por Windows XP en su propia emulación de DOS. La desventaja de esta emulación es que puede ocasionar una gran pérdida de rendimiento.

Para ayudar en la virtualización, VT y Pacifica han añadido un nuevo nivel de privilegio más allá del Ring 0. Ambos han añadido nueve instrucciones más que sólo pueden ser ejecutadas en el *Ring -1*, y que serán usadas por el hypervisor. De este modo el SO no deberá ser modificado, y las penalizaciones de prestaciones se reducirán. Sin embargo, no se eliminarán completamente, ya que cada SO debe estar totalmente aislado de los otros, y el hypervisor gestionará los recursos para que esto sea así.

La memoria ha sido parcialmente virtualizada desde el 386 en el sentido de que el sistema operativo y un controlador hardware gestionan la RAM destinada a las aplicaciones. La CPU de AMD tiene en este punto una ventaja, pues incluye un gestor de memoria, de modo que Pacifica simplemente tiene que reusarla. En contraste, las CPUs de Intel controlan la memoria en un chip separado no soportado por la tecnología VT, implicando que el hypervisor será el encargado de llevar a cabo este trabajo.

En este momento, la virtualización de dispositivos de Entrada/Salida requiere que los drivers se ejecuten en el hypervisor, por lo que se consideran como drivers virtuales. Futuras versiones de Pacifica y VT eliminarán estos drivers del hypervisor, permitiendo que los SOs invitados se comuniquen directamente con el hardware.

La virtualización puede ayudar a proteger los sistemas operativos contra los fallos o vulnerabilidades, pero conlleva un cambio de concepto tanto en la seguridad como en la estabilidad. El sistema entero es tan bueno en ambas características como lo es el hypervisor. Hasta ahora, los hypervisores han sido muy robustos, y no se han publicado problemas de vulnerabilidades de la mayoría de los productos desarrollados por VMware,

algo que no es habitual en los frecuentes parches requeridos por otros programas. Esto es debido principalmente a que un hypervisor puede ser mucho más pequeño que un sistema operativo entero, lo cual facilita enormemente auditar su seguridad.

Pero VT y Pacifica pueden introducir nuevas vulnerabilidades, especialmente para usuario que no quieren la capacidad de las nuevas máquinas virtuales. Un simple ataque sobre una máquina ejecutando un único sistema operativo no virtualizado no necesitaría ni comprometer el hypervisor, bastaría con que el atacante pudiese introducir un virus o un troyano en Ring -1 que no está siendo utilizado.

Un virus en el anillo -1 sería una nueva variante de rootkit. Como se ejecutaría por encima del sistema operativo y simula el chip x86 exactamente, podría atacar un sistema perfectamente seguro. En realidad, se puede considerar que sería independiente del sistema operativo, el mismo virus podría comprometer cada sistema x86, desde Windows a Solaris. Y sería totalmente imposible detectarlo a nivel de software desde el sistema operativo.

Para protegerse de tales virus, el sistema necesita de un componente hardware que no pueda ser virtualizado. Esto es proporcionado por el "Trusted Platform Module" (TPM), el controvertido chip PKI introducido ya en numerosos PCs. El TPM vigila el hypervisor y otros programas cuando se cargan en memoria, comprobando sus valores hash anteriormente calculados. Una vez se asegura que el hypervisor no ha sido comprometido, es firmado con un certificado digital que puede ser verificado por el sistema operativo virtualizado o por el software de seguridad correspondiente.

En cualquier caso, VT y Pacifica proporcionan ambos una aplicación competente para el TPM [24], incluso para entornos que no necesitan aún máquinas virtuales e hypervisores, como sistemas de sobremesa o portátil, debido a las numerosas ventajas que pueden llegar a ofrecer.

2.3 Virtualización mediante software

2.3.1 Introducción

Tal y como se ha comentado previamente la virtualización mediante software es la aproximación más madura y es ofrecida por numerosos productos. A continuación, se enumerará cuales son las técnicas utilizadas por productos como VMware, Xen o VFe.

Existen grandes familias de productos utilizados para la virtualización. El primero y más conocido es VMware [22], que busca una total virtualización y que permite ciertas posibilidades únicas como ejecutar Windows y Linux en el mismo servidor. Otra aproximación es la paravirtualización, ofrecida por el proyecto de código libre Xen [20], en el que el kernel es modificado, por lo que el sistema operativo es consciente de que esta ejecutándose virtualizado. Por ahora, debido a problemas de licencia, no es posible ejecutar Windows sobre Xen, por lo que VMware mantiene cierta ventaja (aunque técnicamente se ha verificado que es posible).

Otra aproximación, aún en el mundo GNU/Linux, es llevada a cabo por Virtual Iron con su producto VFe [23]. Éste te permite una colección de servidores x86 y alojarlos en cualquier equipo desde una fracción de CPU hasta 16, ejecutando la misma imagen del sistema operativo. VFe requiere modificaciones en el kernel, a cambio de muy altas prestaciones en las comunicaciones entre los servidores interconectados.

Por su parte, Microsoft también cuenta con productos destinados al mercado de la virtualización. Así pues, cuenta con Virtual PC como solución de máquina virtual para sistemas operativos de cliente, o Virtual Server, orientado más a sistemas operativos de servidor.

Por último, se destaca también el producto Virtuozzo, y su versión libre, OpenVZ [25]. Este producto si bien es cierto que no cuenta con la popularidad de los anteriores, con movimientos como la liberalización de su producto está ganando adeptos.

En cada una de estas aproximaciones, la idea es perder un poco en prestaciones o rendimiento, a cambio de ganar en términos de flexibilidad de uso de recursos.

2.3.2 VMware

VMware, como se ha comentado, proporciona una virtualización completa con todas sus ventajas e inconvenientes. Para ello, crea una plataforma virtual de hardware igual en todos los casos, independientemente del hardware real del equipo. La gestión de memoria y de dispositivos de entrada/salida es completamente virtualizada, dando la posibilidad de ejecutar una amplia variedad de sistemas operativos. Por otro lado, esto presenta repercusiones en las prestaciones ofrecidas. Pueden ser especialmente significativas en

aplicaciones que hacen uso especialmente intenso de dispositivos de entrada y salida, o los que carguen demasiado el sistema gestor de memoria del sistema (como aplicaciones que frecuentemente maten y creen nuevos procesos).

Incluso así, Vmware es la primera opción de uso en la mayoría de los data center, por lo que el problema de las prestaciones no debe ser realmente importante. Si el objetivo es crear un entorno donde poder simular servidores lentos o antiguos cuya utilización no se prevea excesivamente alta, Vmware es una opción muy atractiva. Permite la capacidad de mover aquellas aplicaciones a sus correspondientes sistemas en una plataforma consolidada y de un modo que no les afecta. Y como no requiere modificaciones del código fuente del kernel, como les sucede a Xen o Virtual Iron, Vmware no necesita de tantas pruebas para el desarrollo de nuevos entornos.

Sin embargo, existe un caso en el que sí son necesarias las pruebas. Si el sistema invitado no es idéntico al del equipo físico, el entorno para un SO invitado ha de ser modificado en tiempo de ejecución. Esto es porque VMware debe gestionar ciertas instrucciones x86 que implican por sí mismas utilización de memoria. Dado que VMware no modifica el kernel previamente a la compilación, debe ser realizado en el tiempo de ejecución. Si el objetivo es un entorno x86 idéntico al equipo físico, esta situación no se produce. Esto significa que incluso con VMware es necesario una serie de pruebas previas al desarrollo.

Las últimas novedades de Vmware son el soporte de SMP (Symetric MultiProccessing). Permite a una única VM utilizar los cuatro procesadores de un servidor, o tener los procesadores divididos según se necesite. Antiguamente, los SO invitados podían acceder tan solo a una única CPU. Esta capacidad permite a VMware gestionar aplicaciones más grandes, que mueven más cantidad de datos críticos. Debido a esto y a la sofisticación con la que es tratada la gestión de dispositivos de entrada salida, VMware muestra actualmente un nivel de madurez y estabilidad que sus competidores no pueden ofrecer. Además, su trabajo junto con AMD, Dell, HP, Intel, Novell, Red Hat y otros hace que su futuro no pueda ser sino aún más esperanzador.

La capacidad de VMware de migrar las máquinas virtuales entre servidores físicos está suficientemente estudiada e implementada. Esta gestión de software es otra cualidad más que ayuda a mantenerlos en cabeza en el uso en los datacenter. Esto además implica que Intel y AMD están trabajando en hacer que la gestión de VMware de la virtualización sea más sencilla.

2.3.3 Xen

Xen es una máquina virtual de código abierto desarrollada por la Unidad de Cambridge. La meta del diseño es poder ejecutar hasta 100 instancias de Sistemas Operativos con todas sus características, de forma completamente funcional en un equipo sencillo. Xen proporciona aislamiento seguro, control de recursos, garantías de calidad de servicio y migración de máquinas virtuales en vivo. Los sistemas operativos deben ser modificados explícitamente para correr Xen (aunque manteniendo la compatibilidad con aplicaciones de usuario). Esto permite a Xen alcanzar virtualización de alto rendimiento sin un soporte especial de hardware. Mientras que su desarrollo es fundamentalmente para Linux, Xen no está desarrollado de un modo que sea para Linux especificamente. Otras variantes de Unix pueden usar Xen, e incluso existe un port de Windows XP funcionando bajo Xen también.

Xen utiliza la paravirtualización para alcanzar alto rendimiento (es decir, bajas penalizaciones del rendimiento, típicamente alrededor del 2%, con los peores casos de rendimiento rondando el 8%; esto contrasta con las soluciones de emulación que habitualmente sufren penalizaciones de un 20%). Con la paravirtualización, se puede alcanzar alto rendimiento incluso en arquitecturas (x86) que no suelen conseguirlas con técnicas tradicionales de virtualización. A diferencia de las máquinas virtuales tradicionales, que proporcionan entornos basados en software para simular hardware, Xen requiere portar los sistemas operativos para adaptarse al API de Xen. Hasta el momento hay ports para NetBSD, Linux, FreeBSD y Plan9. En 2005 Novell muestra un port de Netware para Xen. Un port de Windows XP fue creado durante el desarrollo inicial de Xen, pero las licencias de Microsoft prohiben su lanzamiento público.

Además, Intel ha realizado modificaciones a Xen para soportar su arquitectura de extensiones VT. Esta tecnología permitirá que sistemas operativos sin modificaciones se ejecuten en máquinas virtuales Xen, si el sistema soporta las extensiones Vanderpool o Pacifica (de Intel y AMD, respectivamente, extensiones para soportar virtualización de forma nativa) Prácticamente, esto significará que habrá una mejora de rendimiento, y que será posible virtualizar Windows sin tener que modificarlo.

Otra característica es que las máquinas virtuales Xen pueden ser migradas en vivo entre equipos físicos sin pararlos. Durante este proceso, la memoria de la máquina virtual es copiada iterativamente al destino sin detener su ejecución. Una parada muy breve alrededor de 60 a 300 milisegundos es necesaria para realizar la sincronización final antes de que la máquina virtual comience a ejecutarse en su destino final. Una tecnología similar es utilizada para suspender las máquinas virtuales a disco y cambiar a otra máquina virtual.

Xen funciona actualmente en sistemas basados en x86. Actualmente se están portando las plataformas x86_64, IA64 y PPC. Los ports de otras plataformas son técnicamente posibles y podrán estar disponibles en el futuro.

Como se ha mencionado, en contraste con Vmware, Xen parte de la modificación y cooperación de los sistemas operativos que se van a virtualizar. Mas concretamente, el sistema operativo consigue acceso a la memoria real, y a dispositivos de entrada salida, evitando los procesos que conducen a la lentitud de los esquemas de virtualización completa. En relación con el modelo de anillos antes comentados, el hypervisor trabaja en el anillo más interior, el 0, dejando el anillo 1 para el sistema operativo invitado. Como las modificaciones del kernel, correspondientes al traspaso de instrucciones privilegiadas a ordenes al hypervisor, son en la memoria y en la gestión de entrada salida, las aplicaciones no necesitan modificaciones para ser ejecutadas en un sistema operativo sobre Xen. Es importante destacar que el acceso a memoria permitido para el sistema invitado es en modo de sólo lectura, siendo necesario el uso del hypervisor para poder escribir en memoria. Vmware sostiene que como se modifica el kernel, las aplicaciones deberían ser probadas de nuevo, pero la propia experiencia demuestra que no es así. Debido a que VMware realiza estos cambios igualmente durante el tiempo de ejecución, las mismas aplicaciones deberían aumentar en sus comportamientos anómalos, pero las pruebas son bastante más difíciles de llevar a cabo.

El argumento de los problemas que pueden acarrear las modificaciones del kernel muy pronto será eliminado. En febrero, el equipo de desarrollo del kernel Linux anunció que las modificaciones para el soporte de Xen serán parte de la distribución del kernel 2.6. Y más aún, Red Hat y Novell han asegurado soporte para Xen, así como sus proveedores Intel, HP y IBM.

Xen evita claramente algunos de los cuellos de botella de VMware, pero aún hay trabajo por hacer para conseguir que la virtualización de Xen sea completamente funcional. Novell, por ejemplo, ya soporta Xen en su distribución Suse Linux Profesional, pero está desarrollando un nuevo conjunto de herramientas de gestión para la virtualización. También intentar añadir soporte a Xen para la "Common Information Model" (CIM), de modo que cualquier consola de gestión compatible con CIM podrá gestionar una máquina virtual Suse Linux.

Como última novedad de Xen, destacar que la versión 3.0 soportará SMP en los sistemas invitados.

2.3.4 Virtual Iron

Mientras VMware y Xen están ocupados intentando copar el mercadado de servidores para muchos sistemas operativos, Virtual Iron intenta ir más allá del entorno del simple servidor yendo hacia un entorno clusterizado. Con VFe 1.0, una sola imagen de SO puede ejecutarse en una fracción de procesador, como normalmente se hace en VMware y Xen. Pero además, también puede ejecutarse en 16 CPUs simultáneamente. Para que esto suceda, Virtual Iron requiere un switch "Topspin Communications InfiniBand" para conectar todos los sistemas participantes del sistema virtual.

Haciendo esto, VFe esencialmente crea una versión extrema de una máquina NUMA. Ello también hace más fácil la tarea de virtualizar redes y subsistemas de almacenamiento, porque todo el proceso de entrada/salida es realizado desde la interfaz Infiniband. El acceso a red es gestionado a través del switch Topspin. Este uso del switch Infiniban que hace Virtual Iron acerca aún más el modelo de "todo virtualizado", como podemos imaginar. Por otro lado, este modelo viene con algunos cambios únicos.

VFe tendrá que manejar un "pool" de memoria que es bastante uniforme. Por ejemplo, es posible que las aplicaciones puedan necesitar acceder a datos en memoria a través del switch InfiniBand, ya que estén situados en la memoria de otros sistemas. El truco es gestionar la memoria correctamente de modo que tales sucesos no sean habituales y que el VM monitor de VFe no gaste mucho tiempo gestionando los procesos para óptimas prestaciones. Aún hay mucho que ganar optimizando el rendimiento de sistemas multiprocesadores con acceso a memoria no uniforme.

Mientras la aproximación al cluster hace que la gestión de memoria de VFe sea bastante más compleja, tiene sus ventajas. La más obvia es la capacidad de aumentar los recursos de cualquier servidor. De este modo, Virtual Iron resuelve el problema de la escalabilidad que ni VMware ni Xen resuelven. Aquellos que necesitan este tipo de escalabilidad pueden a menudo justificar su adquisición. Sin embargo, VFe es demasiado nuevo, así que es difícil de tener una medida precisa de las prestaciones.

La imagen sencilla del SO que hace VFe simplifica algunos problemas que están habitualmente presentes en los cluster. En un cluster estándar, cada nodo ejecuta su propio SO, así que típicamente hay un sistema de ficheros del sistema cluster para asegurar la integridad de los datos para cada miembro. Compañías como Veritas han proporcionado durante mucho tiempo software de gestión de cluster y de sistemas de ficheros. Otro de los problemas que resuelve es la gestión de CPUs propias de los cluster, y las tareas de añadir y quitar recursos, que serán gestionados por el entorno VFe.

2.3.5 Microsoft Virtual PC y Virtual Server

Por su parte, Microsoft ha decidido tener dos productos diferenciados para ofrecer la virtualización. Ambos son productos desarrollados por Connectix, adquiridos posteriormente por Microsoft. Como características fundamentales, Microsoft Virtual Server [27] permite la creación de máquinas virtuales en los sistemas operativos Windows XP y Windows Server 2003. Las últimas funcionalidades añadidas incluyen el soporte de Linux como sistema operativo invitado, soporte SMP (aunque no lo virtualiza), soporte de la arquitectura de 64 bits (aunque no en el sistema operativo invitado). Otra limitación es que soporta hasta 64 sistemas operativos invitados, pero esto hoy en día no parece un factor limitante. Por su parte, la próxima versión hará uso de las nuevas tecnologías de Intel VT y AMD Pacifica. Resaltar también que recientemente, concretamente el 3 de Abril de 2006, Microsoft dio un cambio radical a su modelo de negocio, desarrollando una edición especial de este producto que se pudiera descargar gratuitamente, claro posicionamiento en su lucha por el mercado con Xen y VMware.

Virtual PC [26] es un producto muy popular, principalmente por la familiaridad que conlleva para los usuarios habituales de Microsoft, que ofrece emulación para Apple Mac, y virtualización para los sistemas operativos de Microsoft. Así pues, Virtual PC, al emular el hardware de un PC podría ser utilizado para ejecutar cualquier sistema operativo invitado, pero la experiencia demuestra que puede dar problemas si se utilizan sistemas no definidos específicamente como objetivos en el desarrollo de Virtual PC. Emula un Pentium II con MMX con una tarjeta de vídeo, una tarjeta de sonido y una tarjeta de red. La versión de Virtual PC que se ejecuta en Macintosh usa recompilación dinámica para traducir el código x86 usado por un PC estándar al código equivalente usado por Power PC en un Mac. Esta versión también utiliza llamadas al sistema especiales para aumentar la velocidad de la emulación. La versión de Windows usa también recompilación dinámica, pero sólo para traducir código del modo kernel (ring 0) al modo usuario (ring 3).

2.3.6 Virtuozzo y OpenVZ

Como se ha comentado, Virtuozzo es otra alternativa para la virtualización, desarrollado por SwSoft, y que en sus últimas versiones soporta tanto Windows como linux. Se basa en virtualizar a nivel de sistema operativo, en vez de crear la capa de hardware a utilizar por otras máquinas virtuales, con lo que se consigue mucha más eficiencia. Su objetivo es crear múltiples VPS (Virtual Private Server) aislados, pero que comparten el mismo hardware. Este concepto representa una abstracción mucho más ligera que la virtualización de un sistema operativo entero. Todas las VPS se ejecutan sobre un único

kernel de sistema operativo, que mediante técnicas de virtualización consigue un óptimo uso de los recursos físicos. Este producto es simular a los contenedores de Solaris 10 y las jaulas de BSD, por ejemplo. Al introducir mucho menos overhead que las máquinas virtuales, se pueden ejecutar concurrentemente más VPS que VMs en la misma máquina virtual. Recientemente, han publicado un fork del producto, OpenVZ, basado en licencia GPL, lo que implica que han liberado el código.

Ninguna tecnología promete cambiar la manera en la que se realizan los negocios de los data center como se prevé que lo haga la virtualización.

Es una tecnología más que probada, pero en ciernes en la arquitectura x86. Como uno podría esperar, los riesgos aumentan con las nuevas tecnologías, pero Vmware es una cómoda prueba y Xen tiene el beneficio de toda la comunidad de desarrolladores que lo gestiona.

De este modo el emulador simula todos los componentes hardware necesarios para ejecutar la aplicación. Esto implica, lógicamente, una sobrecarga que hace que sea sensiblemente más lento que cualquier aplicación que se ejecute sobre una máquina virtual.

2.3.7 Otras alternativas

Los productos software que ofrecen la posibilidad de virtualización son muy numerosos. A continuación se enumeran alguna alternativa más:

- Denali [28] utiliza la paravirtualización para proporcionar máquinas virtuales de alto rendimiento en ordenadores x86. La máquina virtual Denali da soporte para Sistemas Operativos mínimamente especializados hacia servicios de Internet. El sistema puede escalar a millares de máquinas virtuales. A diferencia de Xen, Denali no preserva el interfaz binario (ABI), y algunas aplicaciones deben ser recompiladas para que funcionen con las librerías del sistema operativo; en este sentido es similar a Exokernel. En cuanto al objetivo, la principal diferencia es que Xen está diseñado para poder ejecutar un número moderado de sistemas operativos con todas sus características, en vez de un alto número de sistemas ligeros y especializados, como busca Denali.

- Adeos [29] es una Capa de abstracción de hardware que puede ser cargado como un módulo en el kernel Linux. Fue desarrollado para su uso en un kernel de tiempo real, pero no está siendo utilizado con el fin original.

- Qemu [30] es un simulador basado en una máquina virtual, con pocas prestaciones en comparación con los anteriores. Es de código libre, y si se utiliza junto con un

módulo, kqemu, que trabaja en el espacio de memoria del kernel, que es gratis pero no libre, su rendimiento mejora significativamente.

Un problema de la virtualización es que consume recursos. Existen tecnologías que ayudan a minimizar su impacto, como las extensiones Pacífica de AMD, pero actualmente, existe cierto impacto en las soluciones de virtualización. De mayor impacto a menor impacto, por ejemplo, tenemos VMware Workstation, VMware ESX Server y Xen. Otro problema de la virtualización es la estabilidad: si el hypervisor del sistema de virtualización sufre algún problema, éste puede afectar a la totalidad de las máquinas virtuales.

2.3.8 User Mode Linux

Este modo de virtualización está basado en situar al sistema operativo en el espacio de memoria de usuario, es decir, en los anillos exteriores del procesador. Como se ha comentado anteriormente, este tipo de virtualización implica una modificación del kernel del sistema operativo a virtualizar [31].

De este modo se comprende que las máquinas virtuales generadas por este mecanismo no tendrán acceso directo a los recursos físicos del equipo donde se ejecuten, sino que los utilizarán mediante llamadas al sistema a través del kernel del sistema operativo que gestiona el equipo. Por esto, se realiza la distinción entre

- Kernel host, kernel el sistema operativo del equipo sobre el que se desarrolla la simulación, que tiene acceso directo a los recursos físicos del mismo.
- Kernel guest, kernel de la máquina virtual, que accederá a los recursos físicos a través del kernel host. Esto será transparente al usuario.

La modificación del kernel a priori podría parecer compleja o insegura, pero hay que tener en cuenta que el equipo oficial del desarrollo del kernel lo ha integrado. Para la creación de máquinas virtuales necesitaremos como mínimo:

- Un kernel del sistema operativo a simular. Este kernel se compila del mismo modo que cualquier kernel Linux.
- Un sistema de ficheros, llamado root filesystem. Éste será el sistema de ficheros a utilizar por la máquina virtual generada, y donde se incluirá todo el software disponible.
- Utilidades propias del proyecto UML, para llevar a cabo la virtualización y arranque de las máquinas.

Revisando los requisitos, comprobamos que puede existir un inconveniente en la simulación de varias máquinas virtuales, ya que podríamos pensar que cada máquina virtual al tener su propio sistema de ficheros, el espacio en disco del equipo físico sería el factor limitante. Por ello, se implementa en el proyecto UML la técnica de Copy On Write (COW) mediante la cual, todas las máquinas virtuales pueden utilizar el mismo sistema de ficheros, y almacenar en disco sólo los cambios que se produzcan en cada máquina.

En realidad, el factor limitante en este mecanismo es la memoria. Cada máquina virtual tiene asociado un conjunto de direcciones de memoria, y en caso de que el número de máquinas virtuales sea elevado, el rendimiento de las mismas se verá afectado. Para

paliar en la medida de lo posible esta penalización, existe otro proyecto, Separate Kernel Address Space, SKAS, mediante el cual se modifica el gestor de memoria estandar del kernel Linux, optimizándolo para la implementación de estas simulaciones.

Profundicemos en el funcionamiento de este parche para comprender mejor su función. Tradicionalmente, UML sigue el siguiente proceso:

- Cada proceso UML corresponde a un proceso del equipo físico.
- Existe un thread especial, el llamado "tracing thread", que gestiona todas señales recibidas por los procesos, independientemente de que sea el encargado de procesarlas o no, y simplemente se limite a enviarlas al proceso que la espera.
- Este thread también intercepta las llamadas al sistema de los procesos, función que es primordial y para lo que es realmente necesario.

Esto implica los siguientes problemas:

- El kernel UML está presente en el espacio de direcciones de los procesos y por defecto, se puede sobreescribir. Esto es obviamente un problema de seguridad, ya que cualquier proceso tiene acceso de escritura a los datos del kernel. Ejecutar UML en el modo "jaula" soluciona este problema, pero afectando en demasía al rendimiento.
- UML usa señales para controlar su kernel durante una interrupción o llamada al sistema, las cuales son muy lentas y afectan el rendimiento.

En resumen, el parche SKAS obliga al kernel UML a ejecutarse en un espacio de direcciones de memoria completamente distinto del usado por sus procesos. Esto soluciona los problemas de seguridad antes comentado, haciendo que el espacio de direcciones usado en UML sea exactamente igual que el existente en cualquier equipo. Además, proporciona un notable aumento de las prestaciones eliminando las señales utilizadas por cualquier llamada al sistema en UML.

El modo tradicional de UML es conocido como "tt mode" (Tracing Thread), y el nuevo como "skas mode". Otra de las diferencias visibles en el equipo, además de la disminución de carga de la CPU, es el número de procesos asociados al equipo físico, ya que en el modo SKAS, aparecerán sólo 4 procesos, en vez de los múltiples que aparecen en el modo tt :

- El thread del kernel UML, que se ejecuta en un espacio de memoria distinto, ejecuta el código del kernel, y gestiona las interrupciones de los procesos UML.

- El thread del espacio de usuario de UML, que ejecuta los procesos normales de UML, y gestiona los cambios de contexto entre los espacios de memoria del equipo.
- El driver ubd para entrada/salida asíncrona.
- El thread de simulación de escritura SIGIO.

El kernel UML es, pues, un kernel Linux completamente funcional, con su propio "scheduler", utilizando sólo el kernel host para el acceso al hardware. Para su compilación, se puede partir de las fuentes de un kernel genérico, al que es recomendable aplicar un parche, desarrollado por Blaisorblade, para optimizar su rendimiento. Entre otras funcionalidades, este parche incluye la aplicación del proyecto "sysemu", el cual permite modificar el comportamiento habitual de las llamadas al sistema de modo que no se lleguen a ejecutar, mejorando significativamente el rendimiento de la máquina virtual.

Otro de los aspectos que UML ha implementado es el relativo a la simulación de redes. Actualmente, permite simular una red tanto como conectada mediante un switch, como mediante un hub, ambos a partir del mismo paquete de utilidades software, uml_switch. Este software es fundamental para el proyecto UML pues ofrece numerosas posibilidades, como la simulación de varias subredes en el mismo equipo físico, y dotarlas de acceso al exterior a través del equipo físico. Incluso mediante software adicional, como es el desarrollado por el proyecto zebra, se puede llegar a estudiar el comportamiento del protoco OSPF. Este software será estudiado en profundidad más adelante, pues es de especial importancia para el desarrollo de este proyecto.

Comparado con otras tecnologías, es cierto que las prestaciones ofrecidas pueden ser menores que productos como OpenVZ o Xen, por ejemplo, si bien es cierto que los proyectos de virtualización de Intel o AMD previamente comentados pueden ayudar a mejorar notablemente su rendimiento. Además, cuenta con el factor de su licencia, que permite su uso de forma gratuita, lo cual lo hace idóneo para entornos académicos.

2.3.9 Virtual Network User Mode Linux

Se procede a continuación a un análisis detallado de la herramienta Virtual Network User Mode Linux (VNUML) y su estado al comienzo de la realización del proyecto.

VNUML es una herramienta de propósito general desarrollada para la simulación de redes virtuales en un equipo físico. Esta basada en software libre y distribuida bajo licencia GPL, por lo que cualquier usuario puede modificarla para su uso, y ayudar de este modo al aumento de funcionalidades proporcionados por la herramienta. El objetivo de la misma es facilitar al usuario un mecanismo de simulación de redes, mediante el cual el usuario diseña un escenario, constituido por equipos GNU/Linux e interconectados mediante distintas redes, y lo refleja en un XML (eXtended Markup Languaje) siguiendo el DTD (Document Type Definition) proporcionado por VNUML. De este modo se abstrae al usuario de los detalles técnicos de la virtualización [11].

VNUML soporta los siguientes modos de funcionamiento:

- Construcción de la topología de la simulación
- Ejecución y parada de órdenes en las máquinas simuladas
- Destrucción de la topología

De este modo, se automatizan la mayoría de las tareas que se llevan a cabo para cualquier simulación. El funcionamiento de VNUML es el siguiente:

- Mediante un parser se procesa el fichero en el que el usuario ha descrito el escenario a simular. Este fichero ha de ser válido en función del DTD de la versión correspondiente.
- Cuando todos los datos son parte de un árbol DOM, se procesan para la realización de la acción solicitada por el usuario, bien sea la creación de la simulación, la eliminación, o la ejecución de comandos.

Toda la lógica de control del programa está implementada mediante el lenguaje Perl. Esta lógica se puede resumir en los siguientes puntos :

- Procesamiento de las órdenes introducidas por el usuario (crear la simulación, ejecución de los comandos correspondientes, finalización de la misma...)
- Procesamiento del fichero XML en el que se define unívocamente el escenario de la simulación mediante el Parser.
- Simulación de las máquinas propiamente dichas.

La lógica del primer punto, el procesamiento de las órdenes introducidas por el usuario, es implementada en el propio programa vnumlparser.pl sin mayor complejidad. Para ello, se definen los posibles modos a utilizar (modo de construir la topología, de destruirla, de ejecución de comandos...) y se comprueba que la orden introducida por el usuario se puede realizar, finalizando la ejecución en caso contrario (por ejemplo, en caso de solicitar la ejecución de comandos sobre una simulación cuya topología aún no ha sido creada). Este proceso, aunque laborioso, es el más sencillo de los tres.

A continuación, se procede al procesamiento del fichero XML indicado por el usuario. Este fichero XML ha de ser coherente con la versión del DTD públicamente disponible. El procesamiento del mismo se lleva a cabo mediante el módulo XML::DOM, y el método ValParser, para comprobar que además de ser un XML correctamente formado, es válido según el DTD.

Además, y para comprobar que el escenario definido por el usuario es coherente, en cuanto a que las opciones solicitadas son posibles y no se contradicen por ejemplo, se ha desarrollado el módulo CheckSemantics. En este se procesan y establecen condiciones, que en caso de no cumplirse, harán que se aborte la ejecución del programa, indicando el correspondiente error. Existen además módulos implementados para facilitar el procesamiento de la información del fichero, tales como DataHandler.pm o TextManipulation.pm.

Para el último punto, para la creación de máquinas virtuales y las redes ethernet, se utilizará la técnica de User Mode Linux, ya explicada anteriormente.

2.3.10 Casos de uso

Como se ha visto, existen numerosas alternativas destinadas a ofrecer la virtualización o mejorar las funcionalidades. Desarrollos como los de Intel o AMD no hacen sino reforzar la idea de la importancia de la virtualización. Hasta ahora, se había visto relegada a ciertos colectivos. Pero los días de que para probar una nueva aplicación era necesario comprar un nuevo servidor, conseguir espacio de almacenamiento suficiente y probarlo han finalizado. Ahora se utilizan técnicas mucho más flexibles.

Hoy en día, cada aplicación se ejecuta en su propia máquina virtual diseñada en tamaño y prestaciones para la aplicación. Si la aplicación necesita más recursos, la máquina virtual crece para proporcionárselos

Las ventajas de la virtualización son palpables para los proveedores de servicios: es más rentable y manejable tener pocas máquinas grandes, particionadas en múltiples máquinas virtuales (VPS - Virtual Private Server), que tener muchas pequeñas. Por ejemplo, el ahorro en espacio físico en el centro de datos y el ahorro en energía para el

proveedor de servicios puede ser bastante considerable. No sólo los proveedores de servicios, sino cada vez más clientes, pueden sacar partido de la virtualización: ejecutar dos sistemas operativos completamente diferentes sobre el mismo hardware, simultáneamente, sin necesidad de tener que recurrir al arranque dual, o poder seguir utilizando aplicaciones muy antiguas ejecutando el sistema operativo antiguo en una máquina virtual, segregar mejor los recursos físicos de la máquina y distribuirlos para usarlos de forma dedicada (por ejemplo, asignando un procesador dedicado a cada máquina virtual) o de forma más eficiente (como repartir cuatro procesadores reales entre dos máquinas virtuales).

Destaca asimismo el aislamiento e independencia de servicios y contenidos. Se puede utilizar una máquina virtual diferente para ejecutar servicios web, ftp, correo... La ventaja es que un fallo de alguno de los sistemas operativos no repercute en los demás. También es adecuado cuando se heredan sistemas más antiguos y se pueden producir conflictos entre distintas versiones de librerías. Al tener los sistemas aislados e independientes se consigue un alto grado de seguridad (unos procesos no interfieren con otros)

Evidentemente, implican también un ahorro de hardware, pues podemos alojar varios servidores dedicados en un sola máquina, con el aprovechamiento de los recursos de los procesadores antes comentados.

Las máquinas virtuales son usadas a menudo por IBM, HP y otras compañías en sus servidores y ordenadores centrales para abstraer la mayor cantidad de aplicaciones posibles y securizar las aplicaciones poniéndolas en máquinas virtuales diferentes (semejante a una jaula chroot, pero más seguro) Puede ser también utilizada, no solo por razones de seguirdad o funcionamiento, sino también para poder tener arrancados diferentes sistemas operativos en el mismo ordenador. Con la migración de máquinas virtuales en vivo de Xen se puede conseguir hacer balance de carga sin tiempos muertos.

En ejemplos concretos, los casos son cada vez más numerosos. Uno de ellos muy claro donde las ventajas que la virtualización ofrece son muy tenidas en cuenta es en el "hosting". Éste es un servicio proporcionado por empresas a clientes, en el que se le ofrece al cliente un equipo con acceso a internet en el que el usuario puede publicar diferentes servicios, como son servidores http, ftp, smtp... El mantenimiento del mismo es responsabilidad de la empresa, por lo que el usuario queda libre de esta tediosa labor. Pues bien, cada vez son más las empresas que ofrecen servidores virtuales en vez de servidores dedicados, por el ahorro económico que conlleva, al necesitar menos equipos físicos, menos gasto de alimentación eléctricas, las facilidades de administración que conlleva... La lista de empresas que así lo utilizan es interminable, unos ejemplos muy claros podrían ser vpsland, Rimu Hosting, LinuxVDS, ByteMark Hosting...

Otro uso muy claro es de la seguridad de servicios. Es muy recomendable dedicar cada servidor en producción a un sólo servicio, limitando de este modo las amenazas que pueden

materializarse sobre el equipo. Evidentemente, si sigue esta filosofía, el ahorro de usar máquinas virtuales es más que notable. Como una de las características primordiales es el aislamiento entre las mismas, el resultado es el mismo que dedicar equipos físicos enteros a cada servicio. Además, siguiendo en el campo de la seguridad, son una alternativa idónea en la generación de honeypots, máquinas destinadas a ser comprometidas por atacantes en redes telemáticas, para de este modo, estudiar cuál es su comportamiento. Del mismo modo, gracias a VNUML, la creación de honeynets, que no es más que una red de equipos honeypots, se facilita enormemente.

En entornos de producción, son muy tenidos en cuenta pues ofrecen la posibilidad de migración de servicios, algo fundamental para servicios que han de estar continuamente levantados. Además, permiten probar nuevas versiones de software sin la necesidad de adquirir nuevo hardware, y con la seguridad de que el entorno de producción no se verá afectado en absoluto.

Igualmente, en el ámbito académico sus posibilidades son también enormes. Se pueden probar sin problemas los servicios en cada máquina virtual ejecutándose a tal efecto, sin la preocupación de las posibles repercusiones que ello pueda conllevar. Además, el ahorro en hardware es también notable. El ejemplo más claro, está en la propia asignatura de Laboratorio de Redes y Servicios de Telecomunicaciones, donde existen prácticas que utilizan la propia herramienta VNUML para la generación de los escenarios de estudio.

Capítulo 3

CD Autoarrancable

3.1 Introducción

Uno de los mayores problemas con los que se enfrenta la herramienta VNUML es su proceso de instalación. Se ha de remarcar que dicho paso ha sido notablemente simplificado gracias a la ayuda de varios desarrolladores, con el resultado de un script de instalación que automatiza la mayoría de los pasos necesarios. A pesar de esto, hay determinados procedimientos que pueden no resultar triviales para usuarios noveles.

Por otra parte, es necesario para su uso alguna distribución del sistema operativo GNU/Linux, siendo éste un sistema minoritario que la mayoría de los usuarios no conoce lo suficiente para su utilización habitual.

Es por esto que una herramienta como un CD autoejecutable puede ayudar mucho a la utilización y difusión del programa. Ejemplos muy claros de la importancia de este tipo de software para la distribución y expansión de un producto se encuentra sin ir más lejos en la distribución Ubuntu, sin duda la distribución GNU/Linux de mayor crecimiento de número de usuarios en los últimos años.

El CD contará con ciertas características recomendables, aunque no estrictamente necesarias, para la utilización óptima de VNUML, por lo que se podrá probar la herramienta en sus mejores condiciones.

Así pues, el objetivo de esta sección es la explicación del desarrollo seguido para la implementación de un CD autoarrancable para que cualquier usuario pueda probar VNUML sin mayores necesidades que una copia física del CD y un ordenador, independientemente del sistema operativo instalado en el mismo. Concretando algo más los objetivos, se considera necesario que el CD autoarrancable cuente con las siguientes características:

- Posibilidad de ejecutarse en cualquier ordenador.
- Todo el software relativo a VNUML debe estar completamente instalado, con las herramientas necesarias de User-Mode-Linux, alguna versión de un kernel uml, y

un sistema de ficheros *root-filesystem*, para la creación de las máquinas virtuales. Para mejorar las posibilidades de estudio de éstas, los *root-filesystem* contarán con software de servicios web, ftp, smtp...

- Un kernel para la máquina física con los parches recomendados para el uso de User-Mode-Linux.
- Herramientas típicas, necesarias y útiles, para el análisis de sistemas, como analizadores de protocolos.

Cumpliendo todos estos requisitos, se obtendrá una versión del CD muy apropiada para cualquier usuario que quiera introducirse en el uso de la herramienta sin mayores complicaciones.

3.2 Estudio de CDs autoarrancables

A continuación se procede a la explicación de las posibilidades existentes para comenzar el desarrollo del CD desde un nivel teórico, en el que se explicarán las posibilidades actuales, con sus correspondientes ventajas e inconvenientes, y se sentarán las bases para la posterior implementación.

La práctica totalidad de las BIOS de las plataformas x86 soportan CDs autoarrancables. Esto es gracias a la especificación de "El Torito", que es una extensión de la ISO 9660, y que indica cómo ha de ser formateado un CD para que se pueda arrancar directamente desde el mismo. El proceso de las BIOS consiste en buscar un sector de arranque en un CD ISO 9660, y en caso de existir, le asignará un número al dispositivo. Este número indicará si se utiliza emulación de disco duro, emulación de disquete, o no se usa emulación. Mediante la emulación, se permite arrancar desde CD simulando que se arranca desde disquete o disco duro.

Se puede conseguir simplemente mediante una imagen de disquete cuyo tamaño sea exactamente 1.44 MB situado en algún punto de la imagen ISO del CD. En el comienzo de la imagen ISO, existen punteros indicando la posición de la imagen. La BIOS entonces se comportará como si estuviera arrancando desde un disquete. Este método tiene el serio inconveniente de la limitación de tamaño, 1.44 MB. Por otra parte, la emulación del disco duro presenta problemas de compatibilidad en numerosas BIOS, por lo que tampoco es una opción recomendable. Actualmente, no es necesaria la emulación si se usa un gestor de arranque adecuado, como puede ser Isolinux, explicado posteriormente.

La mayoría de los live-CDs existentes en la actualidad están basados en GNU/Linux, pero existen otros basados en MacOS, Mac OS X, BeOs, FreeBSD, o Microsoft Windows, entre otros. Estos CDs utilizan una imagen de sistema de ficheros comprimidos,

que generalmente dobla la capacidad efectiva de almacenamiento, aunque ralentice su arranque.

Suelen incluir además unos scripts de autoconfiguración y de *Plug-and-play*, necesarios para evitar que el usuario se vea obligado a configurar su sistema cada vez que lo utilice, y los hace significativamente más fáciles de utilizar para aquellos que son nuevos en esos sistemas operativos.

Un estudio de las distribuciones live-CD revela que en la actualidad existen principalmente las siguientes alternativas:

- Knoppix [1]. Se podría considerar que constituye un estandar de este campo. Es la base para una gran cantidad de versiones remasterizadas para multitud de tipos de usuarios y entornos.

- Slax [15]. Basada en Slackware Linux. Proporciona numerosos scripts y utilidades para nuevas remasterizaciones.

- HAL-Linux [14]. Esta distribución ofrece numerosas alternativas a las ofrecidas por Knoppix, como el uso de squashfs en vez de cloop.

- GeexBox.[16] Ofrece la posibilidad de dotar al ordenador de gran cantidad de software para convertirlo en un *media center* con la posibilidad de visualizar contenidos en numerosos formatos conectando el ordenador a la TV, sin ningún esfuerzo por parte del usuario.

- CDs educativos. Existen multitud, como Knoppix/Math, Xplora Knoppix, Freeduc, o MAX, la distribución desarrollada por la comunidad de Madrid.

- Geento o Ubuntu. Ambas distribuciones permiten su utilización desde el CD directamente, y su posterior instalación al disco duro en caso de que así lo solicite el usuario.

En nuestro caso, nos centraremos en Knoppix por diversos motivos, entre los que cabe destacar ser una de las distribuciones pioneras en este campo y la multitud de forks existentes basados en ésta, pruebas evidentes de sus prestaciones, y por sus características de reconocimiento de hardware, algo que se considera fundamental para este live-CD. Veremos a continuación qué características técnicas proporciona, en comparación con otras alternativas.

Knoppix es una distribución GNU/Linux basada en la distribución Debian con el entorno gráfico KDE. Es desarrollada principalmente por Klaus Knopper, y es de libre distribución. Fue una de las primeras en implementar un *Live-CD* o CD autoarrancable,

es decir, un sistema operativo almacenado en un CD que puede ser ejecutado desde el mismo, sin instalarse en el sistema operativo, pero que puede ser instalado posteriormente.

Esta distribución es una de las más conocidas del entorno GNU/Linux, debido principalmente a su reconocimiento de hardware, característica diferenciadora fundamental. Permite que pueda ser utilizada en casi cualquier equipo, de manera que puede ser perfectamente utilizada para la familiarización de los usuarios con entornos GNU/Linux. Además, incluye la mayoría del software más utilizado, como la suite openoffice, navegadores como Konqueror o Mozilla, otros procesadores de texto como Abiword, programas de diseño gráfico como The Gimp..., además del software necesario para incluir varios idiomas. Es por esto que es una de las primeras alternativas a utilizar por los usuarios que no conocen este sistema, o para tareas mucho más avanzadas como recuperación del sector de arranque del disco duro, o análisis forense.

Toda esta popularidad, junto con el hecho de que todo el software incluido en el CD es de código libre, ha hecho que numerosas distribuciones la utilicen como base para su propio desarrollo, modificando los programas que incluye Knoppix por defecto. Los ejemplos son muy numerosos, destacando por su número de usuarios Damm Small Linux, Floppix, Gnoppix, Helix, Morphix o Whoppix, entre otras.

Para solventar el problema de la limitada capacidad de espacio que un CD permite, Knoppix utiliza un sistema de compresión de la imagen que permite almacenar hasta 9 Gigabytes de software en un DVD , y hasta 2 en un CD. Cuando un programa es ejecutado, se descomprime automáticamente y se almacena en la RAM. Es por eso que es recomendable tener al menos unos 256 MB de RAM para ejecutar Knoppix sin problemas.

Otro problema que afecta a este tipo de sistemas operativos es guardar los datos de los usuarios, pues en principio, se puede pensar que cualquier dato se perderá al finalizar la sesión y reiniciar el equipo. Hasta la versión 3.8.1 de Knoppix, esta carencia de persistencia de datos se podía solucionar salvando los documentos explícitamente en una partición del disco duro, o en algún sistema de almacenamiento USB, por ejemplo. Esto podría ser desde el directorio /home del usuario, hasta la configuración del escritorio, por ejemplo. Y podría ser cargado al arrancar, recuperando los datos de la última sesión. El script utilizado para la persistencia de datos puede almacenar los mismos con un cifrado AES256, por lo que la seguridad en cuanto a confidencialidad también está asegurada.

Pero desde la versión 3.8.1, Knoppix, además de lo anteriormente comentado, usa el sistema de ficheros UnionFS [17], que permite salvar transparentemente al usuario cualquier fichero nuevo o modificado a cualquier sistema de ficheros donde tenga permisos de escritura, como puede ser la memoria RAM o en el disco duro. Esto implica que el usuario puede modificar el software instalado en el sistema Knoppix, instalando software adicional. De este modo, ahora, se pueden crear nuevos ficheros en el directorio

/home, o modificar ciertos parámetros de configuración del directorio /etc, por ejemplo. Antes, todo el directorio /usr, al ser de sólo lectura, no permitía la instalación de nuevos ejecutables. Varios métodos se idearon para salvar este obstáculo, como instalar los binarios en el directorio del usuario, y añadir éste al PATH. En algunos casos el funcionamiento de los programas instalados era correcto, pero en otros ocurrían diversos problemas.

Con el uso de UnionFS, se puede considerar que todo el sistema es de lectura-escritura. UnionFS mantiene todos los ficheros que han sido modificados (estos se almacenan en la memoria RAM) y los que no se modificaron. Cuando se accede a un fichero modificado, UnionFS apunta a la copia almacenada en la RAM, en vez de a la copia guardada en el CD-ROM. Esto permite instalar programas, tal y como se hace en la distribución Debian. Estos programas se instalarán en sus directorios habituales, y si se comprueba el directorio /ramdisk, se verificará que existe un espejo de la estructura del directorio raíz, o al menos, de aquellos que han sido modificados.

Estudiamos ahora cómo afronta Knoppix la limitación de espacio. En principio, sólo tendríamos alrededor de los 700 MBs para elegir el software que será incluido, pero gracias a sistemas de compresión de ficheros, este tamaño máximo posible es ampliamente sobrepasado. Entre estos sistemas de ficheros, destacan los siguientes:

- squashfs [32]. No está incluido por defecto en el kernel, es necesario un parche o compilarlo como módulo directamente. Diversas comparativas muestran que es el que ofrece más rapidez en las lecturas.

- cramfs [35]. Sistema incluido por defecto en las versiones 2.4 y 2.6 del kernel. Es utilizado principalmente en sistemas embebidos, debido a que está limitado a 256 MB.

- zisofs [33]. Sistema incluido por defecto en las versiones 2.4 y 2.6 del kernel. Comprime utilizando la compresión gzip. Su ratio de compresión es el más bajo.

- cloop [34]. Al igual que squashfs, no está incluido por defecto en el kernel. Es el que obtiene un mejor ratio de compresión.

Evidentemente, de estas 4 opciones queda automáticamente descartado cramfs. También descartamos zisofs por ser el que menor ratio de compresión ofrece. Un estudio más detallado de cloop y squashfs revela que Squashfs comprime sistemas de ficheros, mientras que cloop comprime bloques, independientemente del sistema de ficheros. Esto implica que Squashfs en principio sea más rápido, con el inconveniente de que cloop comprime mejor.

Se decide continuar con cloop en el proceso de remasterizar Knoppix, tal y como viene por defecto. Además, se ha comprobado que squashfs puede provocar ciertos problemas de compatibilidad con UnionFS, lo que refuerza nuestra elección.

3.3 Implementación

Se describe en esta sección cuales han sido los pasos seguidos y sus bases para llevar a cabo la remasterización de Knoppix, dando como resultado un live-CD que incluye la herramienta VNUML. Para la enumeración de las instrucciones concretas y una guía detallada del proceso seguido, se remite al primer apéndice de este documento.

Para remasterizar el CD, los requisitos del sistema son los siguientes:

- 1 GB de memoria libre total, entre la memoria RAM y la memoria swap. Es un requisito del programa de compresión, compress loop (cloop), ya que la imagen entera del CD ha de caber en memoria.

- 3 GB libres en una partición de disco formateada con sistema de ficheros Linux (ext2/3, xfs, etc.) En esta partición se volcará y modificará el contenido del CD, donde se modificará para conseguir que incluya el software deseado.

En caso de no contar con 1GB de memoria, hemos de aumentar la memoria swap o de intercambio. Esta memoria es utilizada por el kernel Linux como una memoria auxiliar, donde guarda las páginas que se encontraban almacenadas en memoria y que ha pasado un tiempo determinado, por defecto 30 segundos, sin usarse. De este modo, en caso de que vuelva a ser necesario traer la página a memoria de nuevo, es mucho más rápido encontrarla en esta partición destinada a tal efecto que en todo el disco duro.

A continuación, hemos de volcar el contenido del CD en la partición del disco duro que hemos habilitado a tal efecto. Para facilitar el proceso de la creación del CD, crearemos dos carpetas distintas:

- La primera, source, donde se guardarán todos los programas que formarán parte de la imagen, y que se comprimirán mediante cloop posteriormente. Como máximo, debe ocupar unos 1,8 GB para asegurarnos que no tendremos problemas de espacio en el CD.

- Tendremos otra carpeta, master, donde se guardará el conjunto de programas comprimido además de utilidades que nos permitirán llevar a cabo el proceso de arranque del CD y el fichero index.html que es cargado por el navegador Konqueror al iniciar la sesión del usuario Knoppix. Es a partir de la carpeta master de la que crearemos la imagen final del CD.

Como es lógico, ambas carpetas deben ser creadas en la partición del disco duro anteriormente comentada, que tendrá mínimo 3 GBs libres. Para conseguir crear y copiar el contenido de las carpetas, hemos de montar dicha partición desde Knoppix, con permisos de lectura y escritura. Es importante indicarle expresamente los permisos para evitar problemas al cambiar de entorno raíz mediante el comando *chroot*. Con esta instrucción, se cambia el directorio que el sistema considera como directorio raíz, y del que dependen el resto de directorios. Nuestro objetivo es hacer que el sistema piense que el directorio raíz pase a ser el directorio source, para de este modo, comenzar a gestionar el software que se incluirá en nuestro CD.

Es posible que en este punto, al intentar cambiar de directorio raíz, nos aparezca un error por pantalla, indicando que no tenemos permisos de acceso al directorio */dev/null*. Este error puede ser producido si no se ha montado la partición con los permisos adecuados. Si ejecutando el comando *mount* devuelve la opción *nodev*, es que no puede acceder al sistema de ficheros montado, con lo que no se puede acceder a */dev/null*, donde algunos scripts redirigen su salida. Si con el simple reintento de montarlo no se soluciona, hemos de indicarlo expresamente, mediante la siguiente instrucción, recordando siempre que hemos de desmontarlo al finalizar la remasterización del CD.

- *mount –bind /dev /mnt/hda1/knx/source/KNOPPIX/dev*

A partir de ahora, todo el software que instalemos o borremos será el perteneciente al directorio *sources*, o lo que es lo mismo, al futuro contenido del CD. Es muy recomendable para la gestión del software utilizar el comando *apt* debido a las facilidades que aporta. Para poder instalar nuevo software es necesario previamente tener configurada la red, en caso de que no se haya configurado automáticamente al arrancar, además de tener montado el directorio */proc*.

La herramienta *apt*, de *Advanced Package Tool*, al igual que el comando *dpkg*, permite gestionar los paquetes Debian. De este modo, se puede instalar o eliminar paquetes resolviendo todas las dependencias, lo cual puede llegar a ser muy complicado en sistemas GNU/Linux. Este comando se informa de los llamados *repositorios de paquetes* acerca de qué paquetes están disponibles, y de qué paquetes depende cada uno de ellos para funcionar correctamente. Estos repositorios se indican en el fichero */etc/apt/sources.list* que en Knoppix incluye numerosas direcciones distintas. Es recomendable, para resolver correctamente dependencias, y tener software fiable y oficial, utilizar tan sólo un repositorio, el estable y oficial del proyecto Debian. Mediante este comando es posible ver también qué paquetes están disponibles. Otra posibilidad es utilizar la dirección *http://apt-get.org*. Incluso es posible simular qué paquetes se instalarían o eliminarían, o si la instrucción que hemos solicitado finalizará correctamente o no.

Un pequeño manual del comando apt sería :

- Si se desea instalar un paquete determinado, el comando a introducir es *apt-get install paquete_ a_ instalar*
- Si desea buscar qué paquetes puede instalar, *apt-cache search .* | sort | less*
- Si se quiere realizar la búsqueda de paquetes por palabras clave, *apt-cache search palabra_ clave*
- Si ha encontrado un paquete que puede resultar interesante, y quiere más información relativa, *apt-cache show nombre_ paquete*
- Debian guarda una copia del paquete descargado. Esto ocupa un espacio en disco que en el caso de remasterizar el CD, es vital. Por ello, una vez instalados todos los paquetes deseados, se debe ejecutar la instrucción *apt-get clean* con lo que se borrarán todas las copias de paquetes descargados.

Existen otras alternativas de gestión de software en Debian, como son dselect, synaptic o aptitude.

Hay que recordar que el objetivo principal es eliminar la mayor cantidad de software posible, pues necesitamos de algo más de 600 MB para instalar los módulos Perl requeridos por vnuml, el *root-filesystem*, y la versión de la herramienta elegida para incluir en el CD. Estas dos últimas tienen que ser instalados a mano, y en el caso del filesystem, ha de ser descomprimido para que pueda ser utilizado. Comentar también que en el proceso de instalación de VNUML, por defecto se copian los ejecutables al PATH /usr/local/bin. En Knoppix, este path no existe, y hubo que crearlo para que la instalación se completase. Como paquetes a instalar, incluir también *uml-utilities* y *gnome-terminal*.

Todo el proceso de elección del software a incluir en el CD puede ser realizado desde el entorno gráfico que proporciona Knoppix. Para ello, no tendríamos más que definir la variable de entorno *DISPLAY* a donde se quiere levantar el entorno gráfico. Para ello, en la mayoría de los casos hay dos posibilidades:

- En caso de no tener ningún entorno gráfico ejecutándose, pues estemos haciendo todo el proceso desde consola, ejecutar

  export DISPLAY=localhost:0.0

- Si por el contrario, ya existe un entorno gráfico en el equipo, y no queremos finalizar la sesión en el mismo se indicará, desde consola

export DISPLAY=localhost:1.0

Es muy útil además para seleccionar cuál es la apariencia que queremos tenga nuestro escritorio cuando se inicie la sesión normal del CD. Por ejemplo, es conveniente modificar el menú de Kde, o quitar algún botón que no queramos que aparezca. El ejemplo más claro es de la suite openoffice. Este conjunto de aplicaciones son las que más espacio ocupan, y debe ser nuestra primera elección como software a eliminar para conseguir espacio donde instalar el software que nosotros queremos. Pues uno de los botones existentes en el escritorio por defecto de Knoppix corresponde a la suite, y la única manera de eliminarlo es directamente desde el entorno gráfico.

El proceso de modificación de la apariencia del escritorio se basa en que las modificaciones de cada usuario se guardan en directorios ocultos situados en el directorio */home/knoppix*. Una vez que el escritorio tenga la apariencia buscada, si salimos de la sesión con la opción logout del menú de KDE, podemos comprobar la existencia de estos directorios. Pero el proceso no acaba aquí; en los scripts de arranque de Knoppix, se indica que en cada inicio con el CD, el directorio */home/knoppix* ha de ser creado, y como ocurre en la mayoría de las distribuciones GNU/Linux, cuando se crea el directorio personal de un usuario, se copia la configuración establecida por defecto en el directorio */etc/skel*. Así pues, nuestro paso siguiente, despues de finalizar la sesión gráfica, ha de ser copiar todo el contenido del directorio personal al directorio */etc/skel*. Por último, borraremos el directorio */home/knoppix*, para que no interfiera con la creación del mismo directorio con el arranque del CD.

Recordar también que al finalizar, hemos de borrar los ficheros *.bash_history* y *.history* del directorio */root* donde quedan almacenados todos los comandos introducidos en nuestro proceso, y desmontar el directorio */proc*.

Para modificar la página mostrada por konqueror al arrancar Knoppix, en nuestro caso se ha copiado el contenido de la página de la web al fichero index.html anteriormente comentado, incluyendo el resto de directorios como */tutorial*, */reference* o */images* y las hojas de estilo css.

Para autoejecutar algunos programas, se puede crear un script y ponerlo en el directorio /etc/rc5.d/ (Esto sólo carga elementos antes de cargar las X)

Por si se quiere modificar, el fichero de fondo de pantalla se encuentra en */cdrom/KNOPPIX/background.gif*, y en caso de no encontrarse ese fichero, cargará */usr/local/lib/knoppix.gif*. Destacar también los scripts */etc/init.d/knoppix-autoconfig*, encargado de gestionar la configuración inicial del sistema, y */etc/init.d/xsession*, para configurar el software relativo al entorno gráfico.

Al crear la imagen comprimida del software a incluir se utilizará el script

create_compressed_fs. Éste ha sido actualizado, por lo que en nuevas versiones se pueden obtener mejores resultados con la opción -b (best) a costa de ralentizarse enormemente. También existen diversas implementaciones para optimizar el rendimiento de este script, ofreciendo la posibilidad de optimizarlo para su compilación en varias CPUs o no necesitar que quepa entero en memoria.

Una vez obtenida la imagen ISO del CD, puede probarse sin necesidad de copiarla en CD si se cuenta con una partición de sobra con al menos 700MB de espacio libre formateada con ext2, ext3 o Vfat, arrancando desde un disquete. Un disquete de arranque buscará una partición con el directorio */KNOPPIX/* en la raíz, y la imagen comprimida */KNOPPIX/KNOPPIX*. En caso de no contar con dicha partición, para probar el funcionamiento de la imagen ISO sin tener que usar un CD, es recomendable usar algún software de simulación, como puede ser VMware, sobre el que se obtienen resultados óptimos, o qemu. Para éste último, tan sólo hay que ejecutar el siguiente comando para poder probarlo.

qemu -m 256 -cdrom knoppix-custom.iso -boot d -user-net

Como se ha comentado anteriormente, para conseguir resultados de prestaciones aceptables con qemu es muy recomendable utilizarlo junto con el módulo que distribuyen, cuya descarga es gratuita. Aún así, las pruebas que se realizaron sobre VMware fueron las que mejores prestaciones revelaron.

La persistencia de datos era a priori uno de los principales puntos de estudio. Era importante conseguir que los usuarios pudiesen guardar los ficheros que definen sus simulaciones o sus resultados para conseguir que la utilización del CD fuese aún más productiva. La solución óptima viene incluida en los scripts por defecto de Knoppix, que permiten guardar el directorio /home en un dispositivo externo, USB o disco duro, para su reutilización en sucesivos arranques de Knoppix. Para ello, los pasos a seguir son los siguientes:

- Desde el menú de Kde, seleccionar *Configuración*, y *Create a persistent Knoppix home directory*
- A continuación nos aparecerá un cuadro de diálogo al que hemos de contestar afirmativamente.
- Elegimos el dispositivo donde crear el directorio.
- En la siguiente cuestión, hemos de responder no, ya que no queremos usar la partición entera, sino simplemente crear el archivo de imagen knoppix.img en la partición.

- Seleccionamos a continuación el tamaño que tendrá este fichero
- Se puede elegir cifrar el fichero, recomendable en entornos de alta seguridad.
- Al finalizar el proceso, se nos muestra en otro cuadro de diálogo qué opción hemos de introducir en la pantalla de inicio para que Knoppix reconozca el directorio. En general, es suficiente con *knoppix home=scan*, o si por ejemplo seleccionamos como dispositivo donde guardar el fichero */dev/hda1*, *knoppix home=/dev/hda1*.

3.4 Kernel y proceso de arranque del CD

Debido a la importancia para este documento de la compilación de un nuevo kernel que incluir en el CD, y como lleva a cabo el CD su proceso de arranque, se le dedica este apartado especial, en el que se explicarán las bases y los pasos a seguir.

Durante el desarrollo del CD, se partió en primer lugar de la versión 3.9, y posteriormente de la versión 4.0, pues fue publicada durante la realización de este proyecto. En la versión 3.9 el kernel incluido es la versión 2.6.11, mientras que en la 4.0 se encuentra el kernel versión 2.6.12.

Cualquier sistema Linux no es autoarrancable aunque tenga un kernel y un sistema de ficheros raíz válidos, es necesario usar un gestor de arranque que sea capaz de cargar y arrancar el kernel Linux. Los posibles gestores de arranque en Linux son los siguientes:

- LiLo [36]: LInux LOader, es un gestor de arranque que no depende de un sistema de ficheros determinado, y puede arrancar las imágenes del kernel de disquetes y de los discos duros.
- GRUB [37]: GRand Unified Bootloader), es un gestor de arranque más potente que el anterior, que puede cargar numerosos sistemas operativos con multitud de opciones, a costa de una mayor complejidad.
- Syslinux [38]: Este gestor de arranque se utiliza para arrancar desde un sistema de ficheros MS-DOS o FAT de Windows. De esta manera, se usa en dispositivos como disquetes y dispositivos USB. Puede ser usado para crear imágenes autoarrancables para CD-ROMs compatibles con el estandar El Torito, pero parece ser que la mayoría de las BIOS presentan problemas en esta situación. Para esto, utiliza una simulación de imagen de disquete, por lo que tiene un espacio limitado a 1,44 MB.
- IsoLinux [39]: Es un gestor de arranque que utiliza los sistemas de ficheros ISO9660 / El Torito. Evita la necesidad de crear una emulación de imagen de disco con espacio limitado Estandar "El Torito", eliminando la limitación de 1,44 MB.

En versiones anteriores, Knoppix utilizaba Syslinux, mientras que las más recientes usan Isolinux. Evidentemente, nuestra elección será Isolinux. Si se quisiera crear una imagen autoarrancable desde un dispositivo USB, el gestor recomendado a utilizar es Syslinux.

Para la correcta configuración del gestor Isolinux, son necesarios los siguientes componentes:

- Un fichero de configuración: isolinux.cfg.
- El ejecutable encargado del proceso de arranque, isolinux.bin.
- El kernel con soporte de imagen inicial de arranque, initrd.
- Una imagen inicial de arranque comprimida, initrd.gz.

El fichero de configuración de Isolinux ha de indicar el nombre del kernel, y mediante la opción *append*, especificar donde se encuentra la imagen. El hecho de que esté comprimida no es importante, pues el kernel se encargará de descomprimirla posteriormente.

Knoppix utiliza como imagen Inicial RAM Disk (initrd) el fichero comprimido (minirt.gz). Éste es en realidad un entorno mínimo que contiene todos los elementos de un sistema de ficheros raíz, y realiza las siguientes tareas:

- Probar la unidad de CD/DVD.
- Buscar la imagen comprimida del CD.
- Montarla para poder acceder a su sistema de ficheros.
- Pasar el proceso de arranque desde el mínimo entorno raíz de minirt.gz al entorno proporcionado por el sistema de ficheros que se acaba de montar.

Un estudio más exhaustivo del fichero minirt.gz nos muestra más información. Para ello, descomprimiremos el fichero, y lo montaremos como si fuera un volumen de ficheros con la opción loopback en un directorio preparado a tal efecto. A continuación, podremos acceder al contenido de la imagen de arranque que estará reflejado en el directorio correspondiente.

Se puede comprobar que en este fichero existen numerosos enlaces simbólicos y directorios, tales como/cdrom, /etc, /modules, /proc y /static. Se comprueba que todos los demás enlazan simbólicamente al directorio KNOPPIX, que es donde se montará el sistema de ficheros comprimido del CD, cuando minirt haya realizado sus funciones.

Los directorios /modules y /static no son directorios habituales:

- /static es el directorio donde se encuentran scripts como init y modutils. Cuando minirt se monta como directorio raíz en el proceso de arranque, el kernel ejecuta el script linuxrc en este directorio raíz, invocando a los scripts antes mencionados.

- /modules incluye los módulos necesarios para el proceso de arranque de Knoppix. Esto incluye el módulo cloop para descomprimir la imagen, los módulos de reconocimiento de hardware del dispositivo CD o DVD o SCSI, y el módulo UnionFS para la gestión del sistema de ficheros, utilizando el propio CD y la RAM.

Cuando se ejecuta el script linuxrc, se busca la imagen del sistema de ficheros comprimida en el medio desde el que se esta arrancando, sea CD-ROM, USB o DVD-ROM. En este punto, el sistema de ficheros completo se encuentra en la RAM, así que debe ser montado para acceder al sistema de ficheros comprimido. Una vez montado, un segundo disco de RAM es creado con un tamaño determinado en función de la RAM disponible, que será usado para los directorios /home y /var. Finalmente, el módulo cloop.ko es cargado en el espacio de memoria del kernel y se descomprime el sistema de ficheros montándolo en /KNOPPIX, iniciando todos los enlaces simbólicos existentes en minirt. Se crean nuevos enlaces simbólicos, y ciertos ficheros importantes del sistema se cargan en el disco RAM para que puedan ser modificados en caso de ser necesario.

Cuando linuxrc finaliza su función, el control es devuelto al proceso init, encargado de gestionar la ejecución de los scripts de arranque rc, entre los que destaca knoppix-autoconfig. Identificado el proceso de arranque, procedemos a explicar cual es el proceso seguido para modificar el kernel.

Estando situados en el entorno *chrooteado*, procedemos a desempaquetar las fuentes del kernel linux, en nuestro caso versión 2.6.13, en el directorio habitual, para aplicar a continuación los parches que consideramos necesarios, más concretamente serán los relativos a SKAS, ya citado anteriormente, y el necesario para que Knoppix pueda utilizar este kernel.

Para el primer parche, el relativo a skas, elegimos la última versión disponible en la web del proyecto. No es necesario en el proceso de configuración seleccionar ninguna opción del menú, con aplicar el parche es suficiente.

En cuanto a la aplicación del parche *knoppix-kernel.patch*, es importante tener presente que ha de ser el último paso, incluso posterior a la configuración de las opciones del kernel. Esto es debido a que las diferentes opciones existentes para la compilación del kernel modifican ficheros que han de ser sobreescritos por este parche, en especial el fichero *setup.c*, para que pueda arrancar correctamente el CD.

En cuanto a la configuración del kernel, las modificaciones con respecto a la configuración por defecto en principio deberían ser mínimas. Esta configuración por

defecto puede ser encontrada en el directorio */boot*. Hay que tener en cuenta que si vamos a compilar una versión del kernel diferente a la que se incluye en el CD, habrá ciertas opciones que habrán cambiado. En nuestro caso, al partir de una versión 2.6.12, y querer compilar una versión 2.6.13, hubo que realizar varias modificaciones para seleccionar nuevos módulos encargados del reconocimiento de dispositivos USB y simulación SCSI, necesarios para que el CD pueda arrancar en entornos como VMware, por ejemplo.

Como regla general, se puede considerar que se puede compilar todas las nuevas opciones como módulo, excepto aquellas que consideremos necesarias para arrancar correctamente. Por ejemplo, para seguir el proceso de instalación ofrecido por VMware, hubo que compilar el soporte del sistema de ficheros ReiserFS dentro del kernel, y no como módulo, pues en caso contrario, el proceso de instalación no finalizaba correctamente, al no reconocer el sistema de ficheros a utilizar.

Recordamos a continuación las opciones disponibles existentes en el Makefile distribuido por el kernel para proceder a su configuración :

- make config, nos preguntará en modo texto por cada opción no incluida expresamente en la configuración copiada del kernel anterior, tales como nuevas posibilidades de configuración. No es muy recomendable.

- make menuconfig, en la que mediante una interfaz ncurses se procede a la configuración. Fue el método seguido en este caso, y el recomendado por ser el más sencillo y que menos problemas presenta.

- make xconfig, para que la interfaz sea mediante las librerías qt o gtk. Para este método, es necesario tener configurado correctamente el entorno gráfico y la variable DISPLAY.

Y las opciones a elegir en cada módulo en cuanto a su compilación son las siguientes:

- No compilar el módulo. Esto implica que la funcionalidad que ofrece no nos es necesaria. En el caso de este CD, al ser una herramienta de carácter universal, no es recomendable seguir esta opción.

- Compilarlo sólo como módulo. Si queremos una funcionalidad, y no nos es estrictamente necesaria para el proceso de arranque. Si se duda, se recomienda indicar esta opción.

- Compilarlo integrado en el kernel. Si queremos esta funcionalidad en el proceso de arranque ésta debe ser nuestra opción. Hay que tener en cuenta que esto aumenta el tamaño del kernel, estando en este caso muy limitado, por lo que no es una opción recomendable salvo en caso de necesidad.

Una vez configurado el kernel tal y como queremos, se procede a su compilación, mediante las instrucciones :

- *make*, para compilar el kernel.
- *make modules_install*, para compilar los módulos y situarlos en su directorio adecuado para su posterior uso.

En este punto, es recomendable buscar en el fichero README con qué versión del compilador gcc se debe compilar el kernel. En el caso de la versión 2.6.13, se experimentaron numerosos problemas si la versión del gcc no era la 2.95. En caso de que la versión no sea la recomendada, se puede cambiar sin mayores problemas borrando el enlace simbólico a donde apunta *gcc*, y creándolo de nuevo haciendo que apunte a la nueva versión.

Otro paso necesario para la compilación del kernel es la creación del enlace simbólico asm, dentro del directorio */usr/include* sin el cuál no podrá compilarse el kernel.

Finalizado el proceso de compilación, copiaremos el kernel, el fichero *System.map* y la configuración en el directorio */boot*, como es habitual.

El siguiente paso es compilar los módulos cloop y UnionFS a partir del código fuente para la compatibilidad con la versión del nuevo kernel. Esta es una medida de protección del kernel, por la cual un módulo sólo puede ser cargado si la versión del kernel y del módulo (llamada *MAGIC_NUMBER*) es idéntica.

Sin duda, el proceso de compilación del módulo cloop fue el que más problemas nos presentó, pero pudo realizarse gracias a las diversas opciones existentes. Para compilar un módulo, existen dos opciones principales en Debian (y en Knoppix, al ser derivada suya):

- Con el proceso habitual de las instrucciones *make* y *make install* desde el directorio de las fuentes de cada módulo.
- Utilizando la opción de Debian *make-kpkg modules_install* desde el directorio */usr/src*. Este comando buscará dentro del directorio *modules* los diversos módulos y procederá a su compilación.

En cuanto a las fuentes del módulo, igualmente se pueden conseguir de dos maneras:

- Las fuentes estandar de cada proyecto
- Las fuentes del proyecto, modificadas por el proyecto Debian para su uso óptimo en esta distribución. En nuestro caso, seleccionamos esta opción, bajando el paquete disponible en los repositorios.

En el proceso de compilación de cloop, es recomendable modificar el Makefile con los datos relativos a nuestro kernel, en función del fichero conf.vars situado en /usr/src/linux. De este modo, se solucionarán algunos de los problemas que nos pueden llegar a ocurrir.

En cuanto al módulo UnionFS, en la compilación por defecto incluye numerosos símbolos y variables de DEBUG, haciendo que su tamaño sea de 5.3 MB. Este tamaño es demasiado grande para nuestros fines, puesto que la imagen initrd está limitada en espacio. Para evitarlo, modificaremos el Makefile, de modo que no incluya información de debug, sustituyendo la línea *UNIONFS_DEBUG_CFLAG = -g* por *UNIONFS_DEBUG_CFLAG = -DNODEBUG*. De este modo, el módulo pasa a ocupar unos 2.5 MB.

Una vez finalizada la compilación de los módulos necesarios, procedemos a la creación de la imagen minirt, la cual pasamos a explicar. Recordamos que el modo de acceder a este fichero es montarlo como volumen de ficheros, y accediendo al directorio correspondiente. En este directorio hemos de copiar, en primer lugar, el kernel compilado, y los módulos necesarios para arrancar. Como vemos, hay una sección de módulos de SCSI o USB que son los encargados de reconocer los dispositivos como USB, necesarios en caso de intentar utilizar un lector CD por USB u otros lectores que se reconocen mediante la simulación SCSI. Todos estos módulos han de ser sobreescritos por los propios generados en nuestra compilación del kernel. En caso de no necesitar alguno, no sería necesario copiarlos, con el consiguiente ahorro de espacio.

Todo el proceso de búsqueda de los módulos y su proceso de carga en memoria propios del arranque del CD se realiza mediante el script linuxrc. Esto nos proporciona una posibilidad más, y es indicar que cargue el módulo situado en otra ruta distinta a la habitual. Esto puede ser útil en caso de que no tengamos suficiente espacio en el imagen minirt para todos los módulos, por lo que se podría situar el módulo en el directorio master, e indicar en el script linuxrc la nueva ruta. Ésta fue la alternativa tomada en la segunda opción, para poder conservar todos los módulos USB y SCSI. Existiría otra posibilidad, y es la creación de una imagen inicial de arranque (minirt) de otro tamaño, indicándolo en el fichero de configuración isolinux.cfg.

Para completar el trabajo, también es conveniente modificar el fichero que se carga la imagen que se muestra al arrancar el CD y elegir las opciones, que se encuentra en el directorio master/boot/isolinux/boot.msg Aprovechando el comportamiento normal del CD, en el que al finalizar el proceso de arranque y carga del escritorio, se ejecuta el navegador konqueror mostrando una página web de introducción a knoppix, se modificó ésta página para que fuese la misma que la existente en la web del proyecto, http://www.dit.upm.es/ vnuml, para facilitar el acceso a la documentación a los usuarios.

Como parte del proceso de actualización del proyecto VNUML, se actualizaron los

ejemplos de la web a la última versión, 1.7, para que los usuarios pudieran comprobar algunas de las posibilidades de la herramienta de simulación.

3.5 Conclusiones

Tras todo este proceso, se ha conseguido un CD autoarrancable, que se puede ejecutar en cualquier ordenador que permite arrancar desde CD-ROM y que cuenta, además de con casi todas las funcionalidades ofrecidas por el Knoppix original, con las siguientes características:

- VNUML, versión 1.6.2
- Versión 2.6.12 del kernel, con el parche skas3-v8 aplicado.
- Uml-utilities, versión
- Root_filesystem 0.3.2
- Toda la documentación de la herramienta, en la versión comentada

Este CD está públicamente disponible en la web del proyecto VNUML. Para poder probarlo , se pueden utilizar herramientas como qemu o VMware. Este último ha sacado recientemente unos *player* que permiten generar una máquina virtual, de libre utilización. Gracias al trabajo de David Fernández, existe un player que permite probar el CD sin necesidad de pagar una licencia. Las instrucciones para utilizar este player y probar el CD de este modo están publicadas en la web.

Como se ha explicado, para la obtención del CD se ha estudiado las diferentes posibilidades existentes, el proceso de arranque de los CDs, sistemas de compresión de ficheros y cómo efectuar la recompilación del kernel con los correspondientes módulos. Con esta sección, se espera haber contribuido a facilitar a un posible lector la tarea de implementar un nuevo CD que incluya nuevas versiones de la herramienta. Hay que tener en cuenta que en caso de que el objetivo sea la creación de una imagen que arranque desde USB, la mayoría de los pasos serían exactamente iguales. Incluso a partir de la imagen ISO para CD, existe un software para Windows XP, *HPUSBFW.exe* que modifica el sistema de ficheros del dispositivo USB, reformateándolo y cambiando la configuración de sus sectores de arranque para que pueda arrancar directamente desde USB. Otra posibilidad sería añadir una partición al disco USB en la que instalar el gestor de arranque GRUB, y conseguir el mismo resultado. En cualquier caso, la adaptación de la imagen para un dispositivo USB a partir del documento no debe presentar mayores complicaciones.

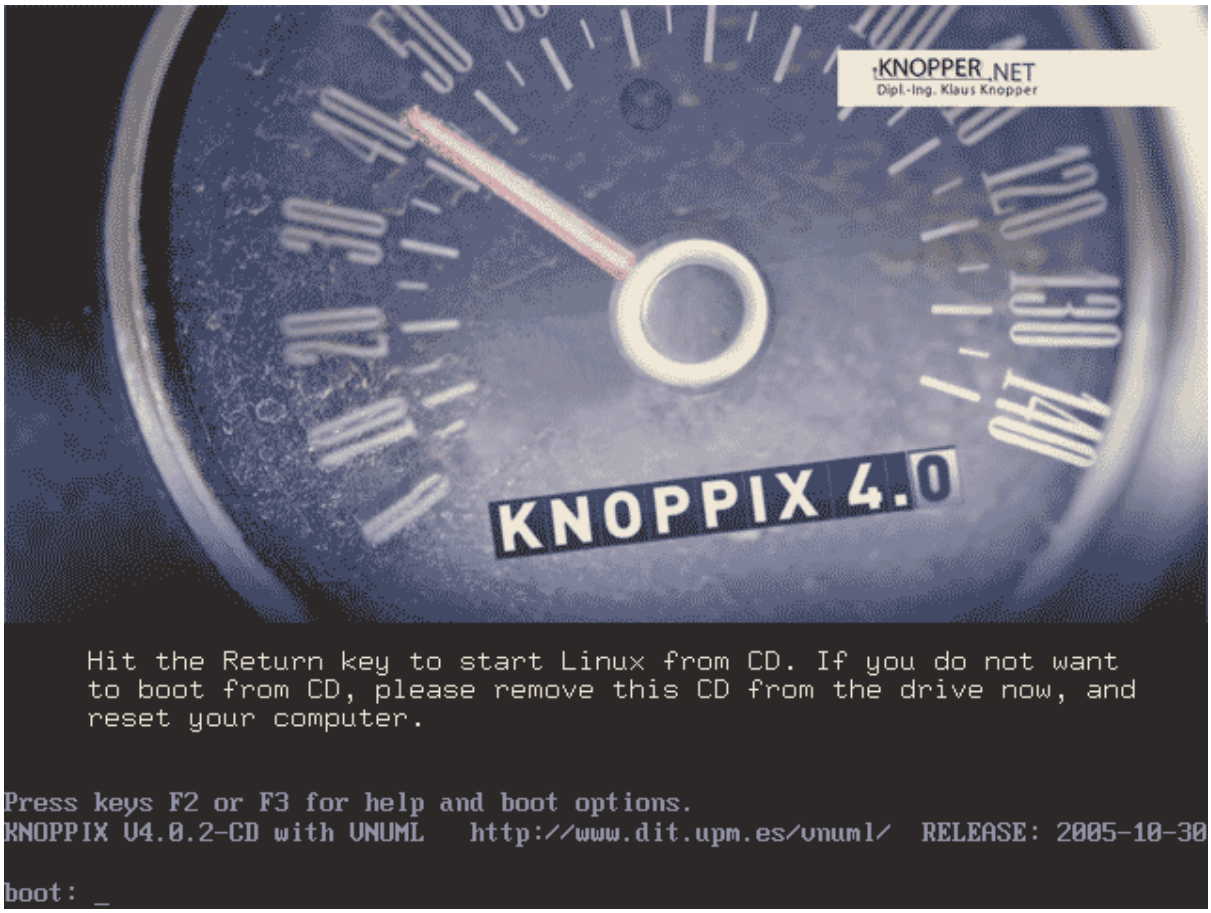

Figura 3.1: Imagen bienvenida CD

Como muestra del CD obtenido, a continuación se incluyen un par de capturas de pantalla del mismo, así la figura 3.1 muestra el mensaje de bienvenida al arrancar el CD, en 3.2 se aprecia la página mostrada al iniciar la sesión del usuario knoppix, mientras que 3.3 muestra una simulación con la topología ya construida desde el CD.

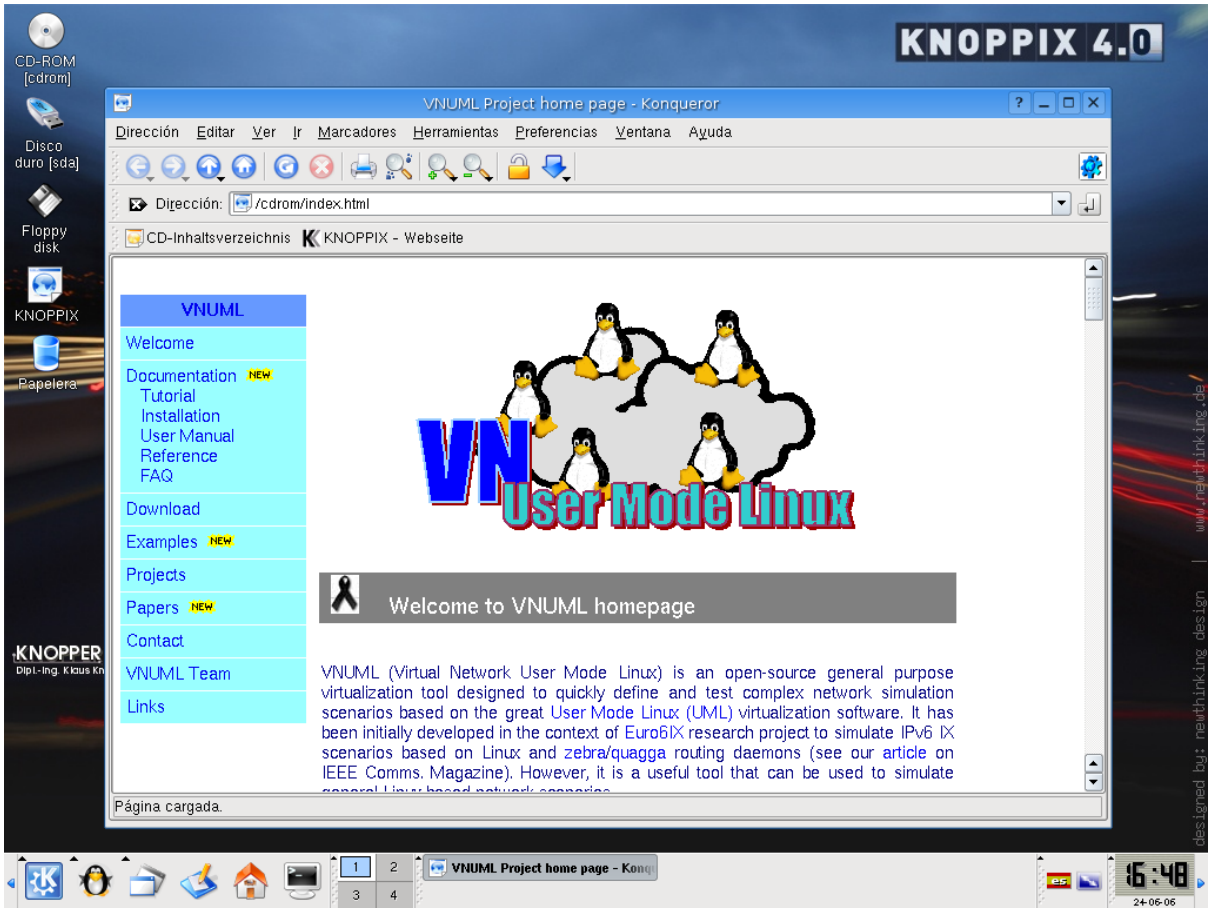

Figura 3.2: Imagen konqueror

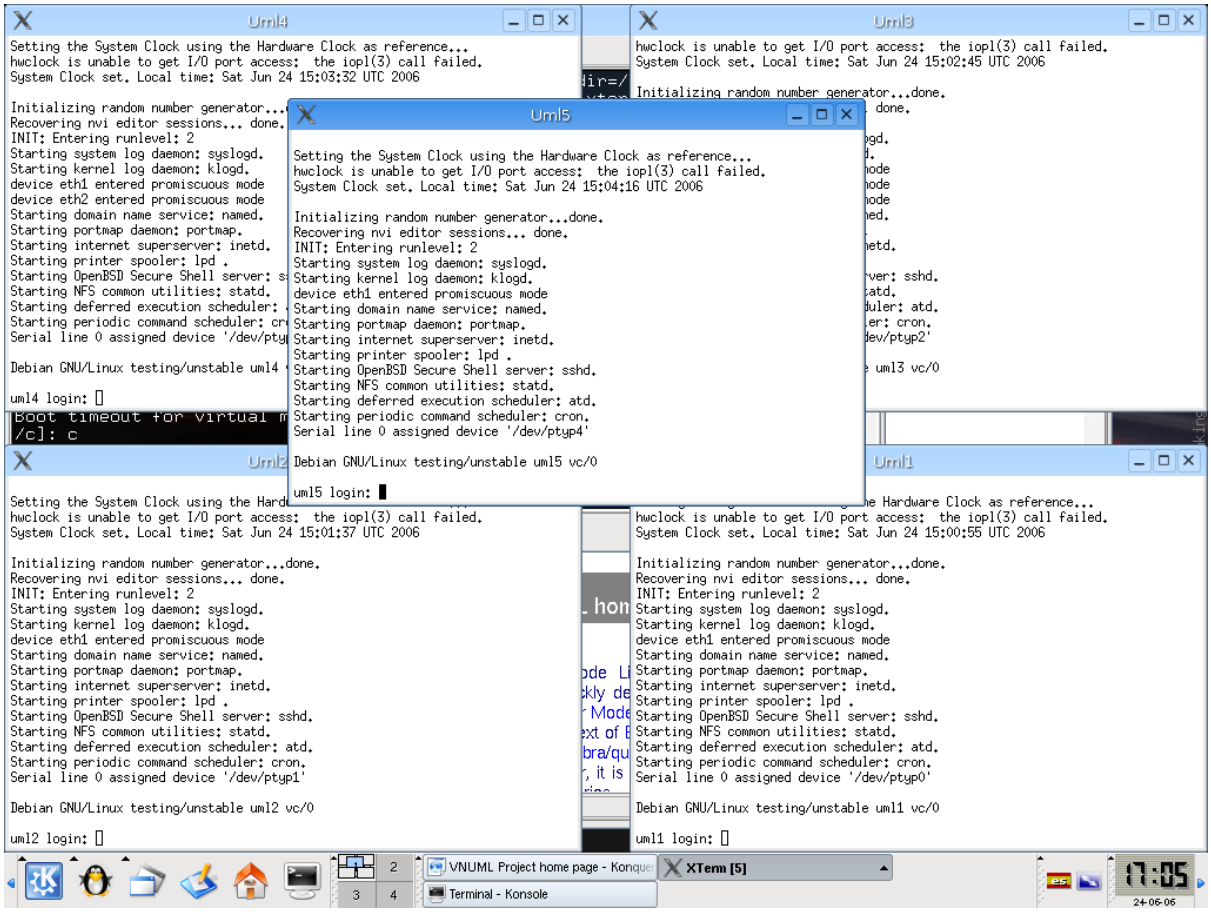

Figura 3.3: Simulación en el CD

Capítulo 4

Analizador de protocolos

4.1 Introducción

Nos centramos a continuación en la herramienta VNUML, con el objetivo de añadir la funcionalidad ya comentada de capturar el tráfico para su posterior estudio.

Para el proceso de creación de máquinas virtuales, VNUML hace uso del software User-Mode-Linux (http://user-mode-linux.sourceforge.net/). Este software es utilizado para ejecutar una máquina virtual Linux que puede tener más recursos virtuales hardware y software que el propio equipo donde se está ejecutando. Esto permite ejecutar software de pruebas, experimentar con nuevos kernels Linux o distribuciones sin ningún riesgo para el equipo físico o su software.

Así pues, para el correcto uso de la herramienta UML son necesarios los siguientes componentes:

- El kernel UML. Este kernel es compilado a partir de las fuentes normales del kernel de Linux, con ciertas particularidades.
- El sistema de ficheros llamado root_filesystem, que utilizará el kernel de Linux como sistema de ficheros raíz, en donde se encuentra el software que se quiere utilizar en la máquina virtual.
- Diversas utilidades y funcionalidades contenidas en el paquete uml_utilities, entre las que se encuentran:
  - port-helper, usado por las consolas que se conectarán a los xterm o ports.
  - tunctl, herramienta de configuración para crear y borrar dispositivos tap.
  - uml_net, herramienta para la automática configuración del dispositivo tap.
  - uml_switch, switch virtual ejecutado en espacio de usuario requerido para la transmisión de paquetes en las redes simuladas.

En este apartado, nos centraremos en el uml_switch, demonio encargado de proporcionar el mecanismo para crear una red virtual, anteriormente conocido como uml_router. Este demonio puede ser utilizado de diversas formas, en función de cual sea la red objetivo a simular, como por ejemplo switch o hub.

El objetivo de este apartado es explicar cómo se ha añadido una nueva funcionalidad a este demonio, un capturador de tráfico o analizador de protocolos. Con la implementación anterior, era posible, mediante herramientas como tcpdump o tethereal capturar el tráfico de ese host determinado. Con esta nueva funcionalidad será posible capturar todo el tráfico de las distintas subredes simuladas.

Para ello, se buscan los siguientes objetivos:

- Posibilidad de capturar el tráfico en un fichero, para su posterior estudio con herramientas como ethereal.
- Posibilidad de capturar el tráfico en tiempo real, con la posibilidad de estudiarlo según se transmiten los paquetes.
- Añadir una funcionalidad de filtrado de paquetes, siguiendo la sintaxis utilizada en la herramienta tcpdump.

Sin duda estos objetivos los hacen muy apropiados para un objetivo puramente académico, pero también sería muy interesante para otros entornos, como puede ser la realización de pruebas de un nuevo software en nuevos sistemas, estudiando cuál es el tráfico normal generado por la nueva aplicación. O por ejemplo, para la creación de entornos honeynets, nos permitirá capturar el tráfico simulando un switch.

Además de estas funcionalidades, se considera interesante aumentar las posibilidades de modo que por ejemplo, se pueda elegir el momento en el que se desea empezar a capturar el tráfico, o cambiar el tráfico capturado a medida que se va produciendo... Por esto, se buscará ofrecer las siguientes funcionalidades:

- Empezar a capturar en cuanto se ejecute el demonio uml_switch, lo cual implica que ha de definirse el tráfico a capturar en el momento de empezar a simular la red.
- Capturar una vez que el uml_switch está siendo ejecutado, con dos posibilidades:
  - Mediante un programa que se comunique con el demonio uml_switch directamente, siguiendo la clásica arquitectura cliente-servidor
  - Mediante un programa que ejecutándose en background actúe de intermediario. Esta opción, pese a ser más compleja, es más interesante, pues permitirá dotar de mayor pues posibilidades a este software para ampliaciones

futuras, además de descargar de mucha parte de la lógica de procesamiento tanto al cliente como al servidor comentados anteriormente.

Para comprender mejor el objetivo buscado, se muestra a continuación un ejemplo de un escenario de VNUML en el que desde el principio de la ejecución se capturará el tráfico entre las dos máquinas virtuales simuladas, con el siguiente comportamiento:

- Se desea capturar todo el tráfico transmitido entre las dos máquinas en tiempo real.
- Adicionalmente, se desea capturar el tráfico con origen y destino en el puerto 25 (tráfico relativo al protocolo SMTP) volcándolo en un fichero para su posterior estudio.

Para conseguir el mismo resultado, se podría haber procedido de la siguiente manera. Se construye la siguiente topología:

Y a continuación, se invocan los siguientes comandos:

- *umlctl direct set-capture_file "/tmp/capture0.cap" tutorial_ capture Net0*, para establecer el fichero de captura.
- *umlctl direct set-capture_ dev "prueba" tutorial_ capture Net0*, para establecer el nombre de la interfaz donde capturar el tiempo real.
- *umlctl direct set-capture_filter "port 25" tutorial_ capture Net0*, para establecer el filtro de captura que se aplicará al volcado de paquetes en el fichero antes definido.
- *umlctl direct start-capture_file tutorial_ capture Net0*, para comenzar la captura en fichero.
- *umlctl direct start-capture_ dev tutorial_ capture Net0*, para comenzar con la captura en la interfaz.

O bien actuar del siguiente modo :

- Ejecutar el demonio *umlswitchd*
- Construimos el escenario mediante vnumlparser.
- Ejecutamos las instrucciones antes enumeradas, sustituyendo la opción *direct* por *daemon*.

Como vemos, existen tres posibilidades distintas. En las siguientes secciones se detallará cómo se han implementado estas funcionalidades, explicando las diferentes posibilidades barajadas, los problemas encontrados y la solución implementada. Además, se dedicará una sección al estudio de las librerías pcap, por su importancia para esta implementación. En caso de querer tener acceso al software, está disponible el código fuente y se distribuye junto con el software VNUML, en la sección *contrib*.

4.2 La librería pcap

Al proceder con el estudio de esta nueva funcionalidad, lo primero que se comprobó, tras observar los productos software que implementan analizadores de tráfico, es que la gran mayoría de ellos utilizaban la librería pcap, *Packet Capture* [6]. Esta librería de código abierto escrita en el lenguaje C proporciona las funcionalidades necesarias, actuando de interfaz entre los programas que procesan los paquetes de red, en espacio de memoria de usuario, y el propio interfaz de red, en espacio de memoria del kernel. Ha demostrado su enorme eficiencia en multitud de programas como tcpdump, ethereal, o su versión para Windows, WinPcap [12]. Así pues, en el desarrollo de esta tarea se añadió como tarea el estudio de esta librería. Nos centraremos en la versión 0.8 de las mismas, por ser la más reciente disponible.

Enumeramos a continuación las funciones más interesantes que se incluyen en la librería para facilitar los programas destinados a la captura de paquetes de red. En primer lugar, proporcionan una serie de funciones que permiten obtener información sobre la configuración de los interfaces de red instaladas. Entre éstas, destacan:

- *char *pcap_lookupdev(char *errbuf)*, que devuelve el primer dispositivo de red válido para captura de paquetes.
- *int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf)*, para obtener la dirección IP y máscara de red del interfaz seleccionado.
- *int pcap_findalldevs(pcap_if_t **alldevsp, char* errbuf)* para obtener todas las interfaces de red que pueden ser abiertas para consultar datos.
- *int pcap_datalink(pcap_t *p)*, que devuelve el tipo de enlace de datos asociado a la interfaz de red.

Una vez obtenida la información relevante, es posible utilizar sus funciones para la captura de paquetes:

- *pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *errbuf)*, para obtener la estructura *pcap_t* necesaria para proceder con la captura de paquetes. Se indicará el nombre de la interfaz de red donde se quiere capturar, y el tiempo en milisegundos que se desea que el kernel agrupe paquetes para entregarlos al programa correspondiente. Esto se explicará más adelante.

- *int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user)*, para capturar y procesar los paquetes

- *int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)*, muy similar a la anterior, pero no finaliza por timeout.

- *u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)*, para leer un único paquete devolviendo su contenido.

Es importante tener presente que los procesos que utilizan las funciones de ésta librería se ejecutan en el espacio de usuario, mientras que la captura de los paquetes tiene lugar en el espacio del kernel, lo cual implica que tiene que existir un mecanismo que actúe de interfaz entre estos dos niveles. Esta característica es fundamental para cuando, por ejemplo, se desean filtrar paquetes. Tiene que existir un filtro en el espacio del Kernel que permita pasar a la zona de usuario (el programa en ejecución) sólo aquellos paquetes cuyos requisitos cumplan las condiciones indicadas. Esta labor es llevada a cabo mediante el denominado Socket Filter.

Cada sistema operativo reescribe su propio sistema de filtrado, pero los más importantes para nuestro caso son los BPF (BSD Packet Filter) [13] y los más recientes LSF (Linux Socket Filter). Los segundos son la implementación de los primeros en sistemas Linux.

El funcionamiento de BPF se basa en dos grandes componentes :

- Network Tap, encargado de recoger los paquetes desde el driver del dispositivo de red y entregárselo a las aplicaciones a las que va destinado.

- Packet Filter, encargado de decidir si el paquete debe ser entregado, y en caso afirmativo, eliminar las cabeceras que no corresponda entregar a la aplicación.

En el funcionamiento normal, sin BPF, el driver envía el paquete a la pila de protocolos del sistema operativo. En cambio, si está activado BPF, será procesado por él.

BPF es el encargado de comparar el paquete con cada uno de los filtros establecidos, pasando una copia de dicho paquete a cada uno de los buffers de las aplicaciones cuyo filtro se ajuste a los contenidos del paquete. En caso de que no exista ninguna coincidencia,

se devuelve el paquete al driver, el cual actuará normalmente, es decir, si el paquete está dirigido a la propia máquina se lo pasará a la pila de protocolos, y en caso contrario lo descartará sin que el paquete haya salido en ningún momento de la zona del kernel.

Puede haber procesos interesados en consultar cada uno de los paquetes de la red, lo que echa por tierra todos los intentos por maximizar el rendimiento ya que tendríamos que traspasar la frontera entre la zona Kernel y el espacio de usuario por cada paquete recibido. Para evitar esto BPF agrupa la información de varios paquetes para luego pasarlos todos de una vez. Para mantener la secuencialidad en que se recibieron los paquetes, BPF añade una marca de tiempo, tamaño y offset a cada uno de los paquetes del grupo.

BPF tiene su propio lenguaje para la programación de filtros, un lenguaje de bajo nivel parecido al ensamblador. De modo que Libpcap implementa un lenguaje mucho más amigable para definir sus filtros. Este lenguaje lógicamente debe ser compilado a BPF compatible antes de ser aplicado. A continuación se detalla cómo programar filtros con el lenguaje de alto nivel desarrollado por Libpcap y popularizado por TCPDUMP que a día de hoy se ha convertido en el estandar para definir filtros de captura.

La expresión que se usa para definir el filtro tiene una serie de primitivas y tres posibles modificadores a las mismas. Esta expresión puede ser verdadera en cuyo caso el paquete se pasa a la zona de usuario o falsa en cuyo caso el paquete se descarta sin llegar a salir en ningún caso de la zona Kernel, tal y como hemos visto en la sección anterior [4].

Los 3 modificadores posibles son:

- tipo Puede ser host, net o port. Indicando respectivamente: máquina (por ejemplo host 192.168.1.1) , una red completa (por ejemplo net 192.168) o un puerto concreto (por ejemplo port 22). Por defecto se asume el tipo host.

- dir Especifica desde o hacia donde se va a mirar el flujo de datos. Tenemos src o dst y podemos combinarlos con or y and. Para el caso de de protocolos punto a punto podemos sustituir por inbound o outbound.Por ejemplo si queremos la dirección de destino 10.10.10.2 y la de origen 192.168.1.2, el filtro será dst 10.10.10.2 and src 192.168.1.2 . Si se quiere que sea la dirección destino 192.168.1.1 o la dirección origen 192.168.1.2 el código será dst 192.168.1.1 or src 192.168.1.2. Pueden seguirse combinando con la ayuda de paréntesis o las palabras or y and. Si no existe se supone src or dst. Por supuesto, esto se puede combinar con los modificadores de tipo anteriores.

- proto En este caso es el protocolo que queremos capturar. puede ser tcp, udp, ip, ether (en este último caso captura tramas a nivel de enlace), arp (peticiones arp), rarp (peticiones reverse-arp),fddi para redes FDDI.

Estas expresiones siempre pueden combinarse con la ayuda de paréntesis y operadores lógicos, Negación (! not), Concatenación (&& and), Alternativa (| | or).

A continuación se detallan las primitivas que pueden usarse. Lo que aparece entre [ ] es opcional, y el | (pipe) significa XOR (o exclusiva).

dst | src host máquina

Verdadero si la dirección destino u origen del paquete coincide con máquina la cual puede ser una dirección IPv4 (o IPv6 si se ha compilado soporte para el mismo), o un nombre resoluble por el DNS.

- ether src | dst | host edir

  Este filtro es cierto si la dirección origen (src), la destino (dst) o cualquiera de las dos(host) coincide con edir. Es obligatorio especificar uno de los tres modificadores.

- gateway máquina

  Verdadero en caso de que el paquete use máquina como gateway. máquina debe estar definida en /etc/ethers y /etc/hosts. Los paquetes capturados por este tipo de filtro, son aquellos cuya dirección Ethernet destino es máquina, pero la dirección IP destino es la de otro sistema.

dst | src port puerto

Este filtro capturará el paquete en caso de que el puerto (udp o tcp) coincida con puerto. El puerto es un valor numérico entre 0-65535 o bien un nombre resoluble a través del /etc/services.

- less longitud

  Verdadero en caso de que el tamaño del paquete sea menor o igual longitud.

- greater longitud

  Verdadero en caso de que el tamaño del paquete sea mayor o igual que longitud.

- ip proto protocolo

  En este caso escucha el protocolo que se le indique. El protocolo puede ser icmp, icmp6, igmp (internet group management protocol), igrp (interior gateway routing protocol), pim (protocol independent multicast), ah (IP Authentication header), esp (encapsulating security payload), udp o tcp. Por comodidad disponemos de los alias tcp, udp e icmp que equivalen a ip proto tcp or ip6 proto tcp, etc...

Concretando dentro de las librerías pcap, se nos ofrecen las siguientes funciones para la implementación de filtros:

- *int pcap compile(pcap_t *p, struct bpf_program *fp,char *str, int optimize, bpf_u_int32 netmask)*

  Esta función se emplea para compilar un programa de filtrado en formato Tcpdump (char* str) en su BPF equivalente (bpf_u_int32 netmask). Es posible que durante la compilación el programa original se modifique para optimizarlo.

  netmask es la máscara de la red local, que puede obtenerse llamando a pcap lookupnet. En caso de que la función devuelva -1, se habrá producido un error, para una descripción más detallada de lo sucedido podemos emplear la función pcap geterr()

- *int pcap setfilter(pcap_t *p, struct bpf_program *fp)*

  Una vez compilado el filtro sólo falta aplicarlo, para ello basta con pasar como parámetro a esta función el resultado de compilar el filtro con pcap compile. En caso de error la función devolverá -1 y puede obtenerse una descripción más detallada del error con pcap_geterr().

A continuación, centrándonos en ethernet, ya que es el protocolo que nos interesa, mostraremos cómo se interpretan los datos. Cuando una aplicación quiere enviar datos a través de una red, antes tiene que añadir las cabeceras de los protocolo que vaya a emplear en la transmisión.

Se puede conseguir una cadena de caracteres con el conjunto de los datos del paquete tal cual, también llamados datos RAW, pero para poder obtener información inteligible, tenemos que hacer la labor que la pila de protocolos hubiera hecho por nosotros, es decir extraer e interpretar las cabeceras añadidas por el remitente.

La primera cabecera que vamos a extraer es la Ethernet, definida en el fichero ethernet.h en /usr/includes/net/.

Del código podemos interpretar que el tipo encargado de contener una cabecera Ethernet se llama ether header y que a grandes rasgos tiene 3 datos interesantes: La dirección ethernet origen (ether shost), la dirección ethernet destino (ether dhost) y el tipo de paquete que porta, que puede ser:

- *ETHERTYPE PUP* Xerox PUP.
- *ETHERTYPE IP* Es un paquete de tipo IP.
- *ETHERTYPE ARP* Es un paquete de tipo ARP (traducción de direcciones ethernet a ip).

- *ETHERTYPE RARP* Es un paquete de tipo RARP (traducción de direcciones ip a ethernet).

En el fichero netinet/ether.h están definidas varias funciones útiles para tratar con direcciones Ethernet, como pueden ser la conversión de una dirección Ethernet de 48bits a texto legible, o de formato texto a una dirección Ethernet de 48bits.

Una vez sabido cómo está estructurada la cabecera Ethernet, podemos extraerla del u char * packet, puesto que sabemos que se corresponde con los 14 primeros bytes (sizeof(struct ether header)), de hecho los primeros 14 bytes siempre corresponderán a una cabecera Ethernet, no así los siguientes que dependerán del valor que tome el campo ether type.

En este caso vamos a suponer que el paquete es de tipo IP (ether type=ETHERTYPE IP), de modo que después de los 14 bytes de la cabecera Ethernet, vamos a encontrar una cabecera IP. La estructura que contendrá la cabecera IP está definida en ip.h (/usr/includes/netinet/),de donde se verifica que está compuesta por los siguientes campos :

- Un campo para la longitud de la cabecera
- un campo para la versión IP
- Un campo que indica el tipo de servicio del paquete IP
- Un campo que contiene la longitud del paquete IP completo
- Un campo que lo identifica unívocamente
- Un campo que indica el offset
- Un campo que indica TTL
- Un campo que indica el protocolo
- Un campo para el checksum
- Y las direcciones IP origen y destino

Se comprueba que existen distintas maneras de organizar los mismos datos.

Sin entrar en detalle en todos los contenidos de la cabecera IP, se hará especial hincapié en el campo protocol ya que nos servirá para determinar cual será la tercera y última cabecera a extraer. Existen multitud de protocolos, aunque los más importantes son ICMP (protocolo=1) TCP (protocol=6) UDP (protocol=17) IPV6 (protocol=41).

En el siguiente ejemplo se muestra cómo llegar a extraer el payload de un paquete TCP (IPheader.protocol=6). Los pasos serán los mismos independientemente del protocolo:

- Extraemos la cabecera Ethernet (6 primeros bytes).
- Consultamos el campo ether type, para saber si es IP.
- Si es IP, consultamos el campo protocol.
- Vamos a /usr/includes/netinet y consultamos cómo es la cabecera para ese protocolo.
- Consultamos el tamaño de esta última cabecera y el payload, si es que lo lleva, va a continuación.

Libpcap nos ofrece también la posibilidad de guardar los datos en un fichero para procesarlos más adelante, (lo que en ingles se llama dump to a file) vamos a ver cuales son las funciones específicas para esta tarea.

- pcap_dumper_t *pcap_dump_open(pcap_t *p, char *fname). Esta función se utiliza para abrir un fichero de salida en el que guardar los datos que vayamos capturando. Si todo va bien la función abrirá el fichero char* fname en modo escritura y nos devolverá un puntero a un descriptor de tipo pcap_dumper_t sobre el que podremos comenzar a volcar datos. En caso de error la función devolverá NULL y podemos ver un error más descriptivo con la función pcap_geterr().
- pcap open_offline(filename, errbuf))

  Esta función sirve para abrir un fichero con paquetes ya guardados en formato TCPDUMP en modo lectura. Los parámetros son muy parecidos a los de la función anterior, fname para la ruta del fichero, NULL en caso de que la función falle, la diferencia es que en este caso para consultar la descripción del error consultaremos errbuf.
- void pcap_dump(u_char *user, struct pcap_pkthdr *h,u char *sp)

  Una vez abierto el fichero salida con pcap dump open(), podemos comenzar a loguear los paquetes con esta función.
- void pcap_dump_close(pcap_dumper_t *p)

  Si hemos terminado con el fichero de salida, podemos cerrarlo llamando a esta función.

Las librerías pcap utilizan unas estructuras de datos propias. Dependiendo del sistema operativo, el fichero pcap.h estará ubicado en un lugar u otro (en Linux por defecto esta en /usr/include/). La estructura pcap_t, que aparece en muchas funciones, es

una estructura opaca para el usuario. Por lo tanto no es necesario saber cuáles son sus contenidos, es suficiente con entender que es donde Libpcap guarda sus variables internas.

Para el procesamiento de estadísticas, existe la llamada a *pcap_stats* que devuelve una estructura con valiosa información estadística, incluyendo el número de paquetes recibidos, el número de paquetes rechazados por algún filtro, y los paquetes rechazados por cada interfaz. En cuanto a los paquetes propiamente dichos, cada uno de ellos va precedido por la cabecera genérica anteriormente explicada, en el caso de la captura simple.

4.3 Implementación

En este apartado procedemos a explicar cómo se ha implementado el capturador de tráfico. Para una mayor claridad, a continuación se exponen los pasos secuenciales que se han seguido en el desarrollo :

- Posibilidad de capturar el tráfico en la subred añadiendo una opción al ejecutable uml_switch.
- Posibilidad de filtrar el tráfico que se desea capturar.
- Posibilidad de capturar el tráfico en tiempo real.
- Posibilidad de modificar el comportamiento del uml_switch estando en ejecución, mandándole órdenes de nuevas capturas.

Para su implementación, en primer lugar se realizará un análisis del código que compone el programa uml_switch. Todos los pasos se han realizado partiendo de la base del software de uml-utilities disponible en el paquete Debian de la versión estable. Un estudio detallado del código revela en primera instancia que los ficheros que tenemos que modificar para añadir la funcionalidad deseada son port.c y uml_switch.c. Sobre ambos ficheros se realizarán todas las modificaciones que a continuación se explican.

4.3.1 Captura de paquetes

En esta sección se explica el desarrollo del primer paso, la captura de paquetes de la subred en un fichero para su posterior estudio.En teoría, este desarrollo consiste simplemente en averiguar en qué función el programa gestiona los paquetes, y escribir los campos de la estructura al fichero indicado por el usuario. La idea fundamental es que el fichero pueda ser leído por programas como ethereal, por lo que se intentará seguir su metodología a la hora de reflejar los paquetes en el destino.

El estudio detallado del código revela que la gestión de los paquetes transmitidos en la subred virtual se produce en la función handle_data del fichero port.c. Cada paquete de la red es reflejado mediante una estructura, llamada packet, del programa.

Evidentemente teníamos dos posibilidades, que son las siguientes:

- Utilizar las librerías pcap, que proporcionan funciones destinadas tal fin.
- Implementar el método por nuestra cuenta.

En este caso, la implementación de nuestro propio método es muy sencillo, ya que sabemos la estructura de datos en la que los programas analizadores de tráfico representan los paquetes capturados. Tan sólo tenemos que escribir cada paquete en el formato adecuado en el fichero, por lo que ésta es la opción elegida. Así pues, para añadir la posibilidad de capturar los paquetes, en la función data_handler añadiremos la llamada a una función desarrollada por nosotros, dumpPacket, pasándole dos valores, el puntero a una estructura packet que hace referencia al paquete que estamos tratando, y su longitud en una variable de tipo entero.

Para la correcta interpretación del fichero por las herramientas basadas en las librerías pcap hay que añadir una cabecera al fichero siguiendo la estructura pmagic, cuyo formato se define en el fichero *pcap.h*:

- Un campo de 4 bytes, el *número mágico*, definido como *0xa1b2c3d4*
- Un campo de 2 bytes indicando el número de versión, *PCAP_VERSION_MAJOR*, consideraremos 2
- Un campo de 2 bytes indicando el número de versión, *PCAP_VERSION_MINOR*, tomaremos 4
- Un offset de 4 bytes para reflejar la diferencia en segundos entre la hora local y UTC, tomaremos 0.
- Un campo de 4 bytes para la precisión de tiempo, que libpcap sobreescribe a 0 siempre, por lo que lo inicializaremos a ese valor.
- Un campo de 4 bytes que refleja la longitud máxima de los paquetes a capturar, el llamado *snapshot length.*
- Por último, un campo de 4 bytes que refleja el tipo de conexión, en nuestro caso definiremos *10 Mb Ethernet*

Así pues, queda de la siguiente manera :

p_head.magic = 0xa1b2c3d4;
p_head.version_major = 2;
p_head.version_minor = 4;
p_head.thiszone = 0;
p_head.sigfigs = 0;
p_head.snaplen = 102400;
p_head.linktype = 1; / ethernet */*

Inmediatamente después podemos escribir los paquetes, cada uno de ellos formado por la cabecera de cada paquete seguido por su campo de datos. Esta cabecera se define en la estructura pcap_pkthdr definido en *pcap.h*, que debe contener:

- Una estructura del tipo *timeval* que especifica el momento en que el paquete ha sido capturado.
- Un número de 32 bits indicando cuantos bytes del campo de los datos del paquete capturado, sin incluir esta cabecera, han sido escritos a fichero.
- Un número de 32 bits que refleja el número de bytes incluía el paquete. Este número puede ser mayor que el anterior, si por ejemplo, se ha definido un *snapshot length* menor que la longitud del paquete.

Todo esto se realiza en la función dumpPacket presente en uml_switch.c. Como se aprecia en el código, para cada paquete se definirá su cabecera mediante la estructura pcap_pkthdr, en la que hay que definir tanto la longitud como el instante de su captura, determinado con la función *gettimeofday* presente en cualquier sistema Linux. A continuación, se escribe en el fichero filedump, definido al invocarse el uml_switch, tanto la cabecera como el paquete en sí.

Para definir tanto el fichero filedump como inicializar la variable log al valor correcto, añadiremos la opción *-f capture_file* a los posibles argumentos que se pueden pasar al uml_switch.

4.3.2 Filtrado de paquetes

Una vez implementado la posibilidad de la captura de tráfico, se pensó que sería muy interesante dotarla con la posibilidad de filtrar los paquetes que se quieren capturar, al igual que se hace con otras herramientas de analizador de protocolos. Es decir, que se pudiera decidir qué paquetes serían capturados en función de la dirección IP origen o destino, o de los puertos o protocolo de cada paquete...

Para implementar esta funcionalidad, habría que añádir una lógica de filtrado al programa uml_switch.c. De nuevo se presentan dos alternativas:

- Implementar el filtro por nuestra cuenta utilizando herramientas como Lex (Legical Analyzer Generator).
- Utilizar la librería libpcap.

En este caso, la alternativa de las librerías pcap era claramente más atractiva, y comenzamos a utilizar sus funciones en el desarrollo del programa.

Hay que tener en cuenta que libpcap nos ofrece las siguientes opciones de captura:

- Capturar directamente de una interfaz, indicando expresamente cuál es.
- Capturar desde fichero.

En este punto, a priori la opción de capturar desde dispositivo podría parecer más atractiva, pero hay que tener en cuenta que al ejecutarse uml_switch no tiene en cuenta, para la gestión de paquetes, ningún interfaz virtual. Por lo que tendremos que utilizar la opción de capturar de fichero. De este modo, la lógica del filtrado consistirá en, cada vez que se procese un paquete de la red, escribirlo en un fichero auxiliar del modo antes comentado, siendo esta operación transparente para el usuario. Inmediatamente después, se aplicará el filtro sobre este fichero, mediante nuestra función *apply_filter* implementada en el mismo fichero, en la que se llama a la función *pcap_open_offline*.

Es importante indicar en cada paquete gestionado por uml_switch que finalice la operación vaciando el buffer de escritura de fichero y cerrándolo, pues en caso contrario, si un usuario abre el fichero sin haber finalizado la simulación, no aparecerán todos los paquetes transmitidos en la red.

4.3.3 Captura de tráfico en tiempo real

Esta funcionalidad es muy interesante para el estudio del tráfico según se produce con herramientas como el ethereal. Un estudio de esta herramienta revela que la única posibilidad de ir capturando el tráfico en vivo es capturando el tráfico en una interfaz o dispositivo de red.

Así pues, el driver TUN/TAP, distribuido en el kernel de Linux, se nos presenta como la opción ideal, pues permite la creación de dispositivos de red en espacio de memoria de usuario, sin tener que modificar el comportamiento del kernel, mucho más complicado.

Adentrándonos un poco más en su funcionamiento, comprobamos que TUN/TAP proporciona las funcionalidades necesarias para la recepción y transmisión de paquetes para programas en espacio de usuario. Se puede ver como un simple enlace punto a punto, o dispositivo ethernet, el cual en vez de recibir paquetes por el medio físico, los recibe del programa, y en vez de enviar los paquetes a través del dispositivo, se los envía a la aplicación.

Cuando un programa abre /dev/net/tun, el driver crea y registra el correspondiente dispositivo de red tunX o tapX. Después de que un programa cierra el mismo fichero, elimina los dispositivos antes registrados y todas las rutas asociadas.

Existen diferencias entre TUN y TAP. El primero está destinado a dispositivos de red virtual de enlace punto a punto. Fue diseñado para permitir el soporte para túneles IP. Proporciona además dos interfaces para aplicaciones a nivel de usuario:

- /dev/tunX - dispositivo de caracteres.
- tunX - interfaz del enlace virtual punto a punto.

Las aplicaciones, de nivel de usuario, pueden escribir tramas IP a /dev/tunX y el kernel recibirá esta trama de la interfaz tunX.

TAP es por el contrario un dispositivo virtual de red Ethernet. Fue diseñado para permitir el soporte en el kernel de túneles Ethernet. Proporciona dos interfaces para aplicaciones a nivel de espacio de usuario:

- /dev/tapX - character device;
- tapX - interfaz virtual Ethernet.

Estas aplicaciones pueden escribir las tramas Ethernet correspondientes a /dev/tapX y el kernel las recibirá del interfaz tapX. Del mismo modo, cada vez que el kernel mande una trapa al dispositivo tapX puede ser leída por la aplicación desde el dispositivo /dev/tapX.

La red virtual puede ser considerada como un simple enlace punto a punto o dispositivo Ethernet, que en vez de recibir paquetes de un dispositivo físico, los recibe de programas a nivel de usuario, al igual que en el caso de mandarlos.

Así pues, la principal diferencia es que TUN trabaja con tramas IP, mientras que TAP la hace a nivel de Ethernet.

También puede haber similitudes entre este driver y BPF. Las diferencias entre ambos es que BFP es un filtro avanzado de paquetes, que puede ser añadido a una interfaz de

red existente, y no proporciona ningún interfaz virtual de red. TUN/TAP proporciona una interfaz virtual, a la que es posible añadir el filtro BPF.

Hay que tener presente que en este caso, todo la lógica de la captura de paquetes estará implementada en el programa utilizado por el usuario, limitándose nuestro objetivo a crear una interfaz en la que volcar el tráfico.

Para su implementación en el código, se añade al método dumpPacket una comprobación de si está definido la interfaz donde volcar el tráfico, y en caso afirmativo, escribir el paquete correspondiente. Para ello, el usuario (o el software vnumlparser.pl) ha de haber ejecutado uml_switch con la opción *-dev nombre_ interfaz*. En este caso, se debe inicializar la interfaz correspondiente, asignándole los flags adecuados, y levantándola para poder empezar a usarla en cualquier momento.

Para asignar los flags al dispositivo, se hace uso de la estructura ifreq, que puede ser usada por descriptores de ficheros asociados a sockets en ioctls para configurar dispositivos de red. Su estructura es la siguiente:

```
struct ifreq {
   char    ifr\_name[IFNAMSIZ];/* Interface name */
   union {
   struct sockaddrifr\_addr;
   struct sockaddrifr\_dstaddr;
   struct sockaddrifr\_broadaddr;
   struct sockaddrifr\_netmask;
   struct sockaddrifr\_hwaddr;
   short   ifr\_flags;
   int    ifr\_ifindex;
   int    ifr\_metric;
   int    ifr\_mtu;
   struct ifmapifr\_map;
   char    ifr\_slave[IFNAMSIZ];
   char    ifr\_newname[IFNAMSIZ];
   char *  ifr\_data;
   };
       };
```

En nuestro caso particular, establecemos los flags del dispositivo indicándole que la interfaz tiene que funcionar en modo promiscuo, lo cual significa que recibirá y procesará todos los paquetes que escuche, aunque en principio no vayan dirigidos a ésta interfaz.

Posteriormente, hará uso de la llamada al sistema ioctl. Esta función se utiliza para

manipular los parámetros subyacentes de ficheros especiales. En general, siguen la estructura

- *int ioctl (int fd, int petición, ...)*

En esta función, el argumento fd tiene que ser un descriptor de fichero abierto. En nuestro caso, mandaremos el descriptor de fichero correspondiente a /dev/net/tun, y le haremos la petición TUNSETIF definida como

- *#define TUNSETIFF_IOW('T', 202, int)*

Es necesario también un método que compruebe que se puede crear la interfaz, para lo cual se verifica la existencia del fichero /dev/net/tun, y que tiene los permisos correctos. En caso afirmativo, inicializa la interfaz como dispositivo TUN (no TAP, lo que implica que no tendrá las cabeceras el protocolo ethernet), y con el flag IFF_NO_PI, se especifica que no se proporcione información de cada paquete (*Packet Information*, para evitar ralentizar el tráfico de la interfaz con exceso de tráfico.

Por último, se asignará el flag IFF_UP a la interfaz, significando esto que todo el proceso ha finalizado correctamente y está preparada para recoger el tráfico del uml_switch.

Debido a la estructura del código, el añadir la posibilidad de filtrar este tráfico es muy complicada, además no se antoja necesaria, pues se puede definir un filtro de captura en la misma herramienta ethereal, con lo que el resultado es el mismo.

4.3.4 Envío de mensajes a una instancia del uml_switch en ejecución

En esta sección estudiaremos cómo implementar un mecanismo de comunicación entre procesos [8]. Enunciamos a continuación cuales son las posibilidades disponibles en los sistemas Unix:

- Señales. Utilizadas para enviar eventos asíncronos a uno o más procesos.

  Mediante esta solución, se implementaría un gestor de señales en uml_switch.c, y cuando se recibiese la señal esperada, se pasaría a efectuar la orden asociada, ya sea capturar el tráfico, o dejar de capturarlo, etc. Este método es el más sencillo de implementar, pero adolece del gran inconveniente de que no se puede pasar información mediante señales. Esto es especialmente importante, pues no sería posible definir un fichero de captura en tiempo de ejecución, por ejemplo.

- Sockets. Esta posibilidad es muy recomendable para ser usada en relaciones cliente-servidor, como puede ser un demonio proporcionando un servicio que crea sockets

en los que escucha, y los clientes se conectan a estos sockets y realizan las peticiones correspondientes. Existen dos familias fundamentales de sockets en Linux, que son las familia AF_UNIX y AF_INET. La primera se utiliza como interfaz entre procesos que se ejecutan en la misma máquina, mientras que la segunda está orientada a procesos que se ejecutan en máquinas distintas. Es por esto que, dado la idea es que los procesos que se van a comunicar, uml_switch y el que crearemos nosotros van a residir en la misma máquina, consideraremos sólo la familia AF_UNIX.

- Tuberías. Se basan en la idea en que un proceso escribe los datos, y el otro proceso los lee. Estos datos son tratados mediante el orden FIFO (first in, first out). Este tipo de tubería no tiene nombre, y es necesario que los dos procesos que se vayan a comunicar deben estar relacionados.

  Existe además la posibilidad de otro tipo de tuberías, llamadas "ficheros especiales FIFO", que en vez de ser anónimas y con el fin de de ser utilizadas para una conexión temporal, tienen un nombre como cualquier otro fichero.

- Además de los anteriores, Linux soporta tres tipos de mecanismos, propios de System V [9]. Estos son la cola de mensajes, los semáforos y la memoria compartida. Estos mecanismos tienen los mismos modos de autenticación, y es que los procesos pueden acceder a los recursos sólo pasando un único identificador de referencia al kernel a través de las llamadas al sistema. Estos métodos son:

  - Semáforos, en su forma más simple consiste en una posición en memoria cuyo valor puede ser comprobado y modificado por más de un proceso. En función de este valor, el proceso podrá pasar a ejecutarse o a estado de "sleep".
  - Colas de mensajes, que permiten que uno o mas procesos puedan escribir mensajes que igualmente serán leídos por uno o más procesos. Linux mantiene una cola de mensajes, el vector msgque.
  - Memoria compartida, permite que varios procesos puedan comunicarse vía memoria que aparece en todos los espacios virtuales de direcciones. Una vez que la memoria está siendo compartida, no se comprueba el uso que los procesos hacen de la misma.

Al final, se han utilizado dos de los mecanismos posibles, como son las tuberías con nombre, y los sockets locales. Las razones para usarlos es que los procesos no estarán interrelacionados como padres e hijos, lo cual descarta el uso de tuberías sin nombre, y van a estar situados en el mismo equipo, lo cual desaconseja, aunque no imposibilita, el uso de sockets de la familia AF_INET, más indicados para su uso en la comunicación

de procesos situados en equipos distintos. El utilizar los dos métodos es principalmente debido a razones didácticas, pues comprobaciones posteriores entre ambos métodos no reflejan diferencias apreciables de rendimiento [5].

Para ofrecer un mayor número de posibilidades, se decidió efectuar el envió de mensajes de dos maneras distintas:

- Con un comando directamente al uml_switch indicado. Esta solución implica que cada instancia del uml_switch en ejecución ha de estar definido unívocamente, y para ello, la alternativa idónea era identificarlos mediante el nombre de la simulación y la red que simula.

- Establecer la comunicación entre el comando anterior y uml_switch a través de un intermediario. Este método es con diferencia más complicado que el anterior, pero resuelve algunos problemas que plantean la anterior alternativa, además de que puede ofrecer más posibilidades de uso.

Así pues, está claro que la comunicación siempre se realizará siguiendo la arquitectura cliente servidor, con un proceso cliente mandado peticiones al proceso servidor. En el caso del proceso umlswitchd, funcionará tanto como cliente como servidor, en función de qué comunicación consideremos. Trataremos a continuación cómo han sido implementadas cada una de las alternativas.

Comunicación directa

El objetivo de este apartado es implementar un proceso que nos permita mandar órdenes a una instancia de uml_switch en ejecución. Se persigue poder indicar, por ejemplo, en qué momento o en qué fichero comenzar a capturar, qué filtro aplicar...

Debido a su funcionalidad de controlar y modificar el comportamiento del programa uml_switch, éste proceso recibirá el nombre de umlctl.

Para la comunicación entre los procesos en este caso se empleará sockets de la familia AF_UNIX. En este caso, el rol de cliente está perfectamente identificado con el programa umlctl, mientras que el servidor que espera peticiones es el proceso uml_switch. Pero esto plantea un problema, y es que hay que definir un fichero concreto, que actuará de socket, para cada uno de éstos. La solución más pausible era determinarlo a partir del nombre de la red, que ha de ser unívoca para cada equipo, ya que vnumlparser no permite la ejecución de dos simulaciones concurrentes en las que existan dos redes con el mismo nombre. De este modo, el fichero asociado al socket en el que los uml_switch escucharán peticiones se encuentra en el directorio */tmp*, debido a su carácter eminentemente temporal, y el nombre del fichero estará formado por la concatenación del nombre de la red con la cadena *.umlctl* quedando de este modo algo como */tmp/red.umlctl*

Se implementará un método encargado de inicializar los sockets a utilizar para escuchar las órdenes, al que se le pasará la ruta completa del fichero en el sistema asociado al socket, y se devolverá el descriptor de fichero que hará referencia al socket, y que se usará en el resto del programa. El socket será de la familia AF_UNIX, como ya se comentó anteriormente. Además, los sockets pueden ser de uno de los dos tipos siguientes:

- *SOCK_STREAM* , especifica que se usarán conexiones TCP.
- *SOCK_DGRAM*, estos sockets no establecen y mantienen una comunicación, lo que implica que algún paquete se puede perder o llegar fuera de secuencia, y al tener menos cabeceras son más rápidos.

Puesto que las comunicaciones se realizarán en la misma máquina, la probabilidad de pérdida de paquetes es mínima, por lo que se considera más conveniente elegir el tipo SOCK_STREAM para el flujo de datos. El socket cliente ha de ser del mismo tipo para que la conexión entre ambos pueda establecerse.

A través de estos socket se comunicarán los procesos. El procedimiento a seguir es sencillo; se enviarán desde umlctl la petición correspondiente, y la comunicación, para controlar posibles errores por pérdida de comunicación, se define síncrona, de modo que se procederá esperar una respuesta durante un segundo. Si no se recibe la respuesta, una vez pasado el tiempo determinado, se entiende que la petición no se ha podido realizar.

Concretando, el proceso umlctl tendrá las siguientes partes claramente diferenciadas:

- Por un lado, procesará las opciones introducidas por el usuario al invocar el programa, y creará el mensaje que será enviado al uml_switch.
- A través de la lógica de sockets, se realizará la comunicación, enviando el mensaje, y se esperará una respuesta, que será *Ok* en caso de que la operación se haya efectuado correctamente, o *Nok* en caso contrario. En caso de no obtener respuesta en un tiempo determinado, se considerará igualmente que no se ha podido realizar la operación.
- Por último, se mostrará por pantalla al usuario un mensaje de confirmación si la operación se ha efectuado correctamente, o un mensaje de error en caso contrario.

Se tiene especial cuidado en todo el programa en controlar situaciones inesperadas o anómalas, que pueden producir que el proceso no finalice correctamente, como pueden ser introducir opciones no contempladas, nombres de simulación o de red deliberadamente largos que puedan desembocar en violaciones de segmento...

Para el proceso de esperar durante un tiempo determinado la respuesta en todos los procesos, es recomendable implementar una función encargada de esta lógica de procesamiento, evitando la espera indefinida. Éste método se puede implementar fácilmente mediante la función *select()*. Esta función es muy útil en casos en los que un servidor está esperando varias conexiones a la vez que mantiene conexiones ya establecidas. Se define de la siguiente manera:

- *int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);*

Esta función permite averiguar qué descriptores de los que están abiertos en el sistema están listos para leer, cuales para escribir, y cuales han generado excepciones. numfds ha de tener el valor del descriptor de fichero más alto más 1. Tal y como lo invocamos en nuestro caso, devolverá en la variable readfs el descriptor de fichero que está listo para leer, en caso de que haya alguno dentro del timeout. Definiendo las variables writefds y exceptfds a NULL se indica que no esperamos eventos de estos tipos.

Para el correcto uso de la función select, es recomendable establecer que el grupo de descriptores de ficheros de los que esperamos respuesta es sólo el descriptor de lectura. De este modo, es conveniente usar las instrucciones siguientes:

- *FD_ZERO(&fds);*, para borrar el conjunto de descriptores de fichero.
- *FD_SET(s, &fds);*, para posteriormente añadir el descriptor del que estamos esperando la lectura.

Por su parte, la estructura *struct timeval* queda definida con los siguientes campos

- Una variable entera, que especifica los segundos.
- Otra variable entera para especificar los microsegundos.

De este modo, queda perfectamente definida la función de lectura durante un tiempo determinado, evitando la espera indefinida de la respuesta de algunas peticiones.

Para llevar a cabo esta funcionalidad, se ha de cambiar el comportamiento original del uml_switch indicándole que pase a escuchar peticiones en el socket correspondiente. El fichero utilizado por este socket se crea de la misma manera que en el caso anterior, en función de lo introducido por el usuario en la opción

- *-simulation_name sim_name net*

Una vez creado el socket, comienza a escuchar peticiones. En el momento que le llega algún mensaje lo procesa. En caso de que la petición se realice correctamente, se manejan dos posibilidades:

- Si era una petición de información, como puede ser solicitar el nombre del fichero donde se captura el tráfico, se envía el contenido solicitado.
- En cualquier otro caso, se envía una confirmación de éxito mediante el mensaje *Ok*.

Si por el contrario, la petición no puede realizarse, se enviará el mensaje "Nok".

Para la implementación de la gestión de escuchar peticiones, debido a la estructura del programa, la mejor solución era aprovechar su bucle en el que espera cualquier petición mediante la función poll, y añadir ahí la nuestra.

Esta función poll es muy adecuada para gestionar descriptores de ficheros de sockets o tuberías. Se basa en esperar eventos en descriptores. Se define del siguiente modo:

- *int poll(struct pollfd *ufds, unsigned int nfds, int timeout);*

Donde se considera que timeout son milisegundos (un valor negativo significa un timeout infinito), y ndfs es un array de estructuras del tipo *struct pollfd*, caracterizada por

- Una variable entera correspondiente a un descriptor de fichero abierto.
- El campo *events*, parámetro de entrada, una máscara que especifica en qué eventos está interesada la aplicación.
- El último campo, *revents*, es un parámetro definido por el kernel.

Cuando se utiliza esta función, el kernel espera a que alguno de los eventos indicados en event ocurran durante los milisegundos indicados en timeout.

En cuanto a los mensajes intercambiados entre los procesos, se considera que las opciones básicas que ha de implementar el programa son las siguientes:

- Establecer el fichero donde capturar los paquetes.
- Definir el nombre de la interfaz donde queremos capturar paquetes en vivo.
- Indicar cuál es la expresión del filtro a aplicar.
- Comenzar la captura de paquetes al fichero.
- Comenzar la captura de paquetes a la interfaz.
- Finalizar la captura de paquetes al fichero.
- Finalizar la captura de paquetes a la interfaz.

- Preguntar cuál es el fichero, si está definido, dónde se capturan los paquetes.
- Preguntar cuál es la interfaz asociada, si está definida, dónde se capturan los paquetes.
- Preguntar cuál es la expresión del filtro que se aplica en la captura de paquetes en fichero.

Sabiendo que estas son todas las posibilidades, crearemos los mensajes a transmitir y procesar en función de esta información.

Se ofrece además la posibilidad de cambiar el fichero a través del cual establecer la comunicación introduciendo en el proceso uml_switch la opción "-umlctl_file file".

Comunicación a través de un intermediario

El objetivo de este apartado es implementar la comunicación entre los comandos con otra lógica distinta, y es usando un intermediario. Esto soluciona el problema de qué fichero utilizar para las tuberías con nombre, y ofrece bastantes más posibilidades, a costa de una mayor complejidad.

Así pues, se desería implementar un programa cuyas características sean las siguientes:

- Que esté en ejecución constantemente, de modo que cuando un nuevo uml_switch se ejecute, se lo comunique y se establezca un canal de comunicación entre ambos.
- Que escuche permanentemente posibles peticiones del comando umlctl, de modo que sea el que gestione la respuesta.
- Que ofrezca nuevas posibilidades, como puede ser mostrar todos los uml_switch en ejecución en el equipo, con los parámetros (como ficheros de captura, filtros a aplicar, interfaz de captura...) que tengan definidos, para que un usuario pueda consultarlos.

Debido a las similitudes del proceso que vamos a implementar con un demonio, lo llamaremos umlswitchd.

Otra cualidad interesante, ya que este proceso debe estar continuamente en ejecución, es conveniente que se haga en background para evitar que una consola quede inutilizada al invocarlo. Posteriormente se explicará cómo se implementa.

Profundizando en la lógica que seguirá este proceso, debe cumplir los siguientes requisitos:

- Escuchará peticiones del proceso umlctl, y las enviará al uml_switch correspondiente.

- Todos los uml_switch que se ejecuten se comunicarán con este proceso, enviándole el nombre de la simulación y la red. Umlswitchd comprobará que es una simulación no existente en su lista, y en este caso, se envía al uml_switch cuál será el nombre de la tubería particular por la que se comunicarán uml_switch y umlswitchd.

- En caso de que una simulación termine, así se lo hará saber a umlswitchd, enviando el mensaje correspondiente. Éste se encargará de eliminarla de la lista de instancias que se ejecutan en el equipo.

Procedemos a continuación a explicar cómo se ha implementado todo el proceso. En primer lugar, indicar que se ha definido que la comunicación entre el proceso umlctl y umlswitchd se realizará a través de tuberías, y la comunicación entre uml_switch y umlswitchd a través de sockets locales asociados a ficheros.

Puesto que es de suponer que en cada momento habrá como máximo una instancia del proceso umlctl, la comunicación entre éste y umlswitchd se realizará siempre sobre las mismas tuberías. En cambio, para las instancias de uml_switch, este método no funcionaría, y se implementan las siguientes posibilidades:

- Todos los uml_switch al iniciarse, se comunican con umlswitchd a través de la misma tubería, y por la que se devuelve cuál será el nombre del fichero asociado al socket donde escuchará peticiones ese uml_switch.

- Y el resto de comunicaciones entre ambos procesos se realizarán a través del socket particular de cada instancia.

Se observaron problemas de permisos en la creación de las tuberías. Por esto, es conveniente el uso de la función umask, para establecer que los permisos de los ficheros creados serán 777. Esto es para evitar problemas de permisos, y al ser ficheros especiales, no supone ningún problema en el ámbito de seguridad. Con la función mknod se inicializa el fichero con ruta definida en uml_ctl_file_recv como una tubería de sólo lectura no bloqueante. Esto último es especialmente importante, pues si no se define de este modo, el programa esperará hasta que se introduzcan los datos en la tubería, y esto causaría numerosos problemas. Además, para la espera de datos en sockets o tuberías, se ha implementado el método antes descrito. Finalmente, esta tubería es referenciada para su posterior uso mediante el descriptor de ficheros umlctl_recv.

Evidentemente, umlswitchd debe mantener una lista de las instancias uml_switch ejecutándose en el equipo. Ésta lista tiene un tamaño dinámico, por lo que se decide implementarla mediante una lista de nodos, en la que cada elemento estará definido por:

- El nombre asociado al nodo, compuesto por el nombre de la simulación concatenado con el nombre de la subred.

- El fichero asociado al nodo, donde el uml_switch escuchará peticiones.
- Una referencia al siguiente nodo de la lista.

Es fundamental la correcta implementación de esta funcionalidad, para evitar una incorrecta gestión de memoria. Cuando se reciba la señal de que un nuevo proceso uml_switch se comienza a ejecutar, en el mensaje correspondiente se enviará el nombre de la simulación y la red, que concatenados, determinarán el parámetro name de la estructura nodo. A continuación, se generará un nombre de fichero aleatorio, con números y letras, que se ubicará en el directorio /tmp y que será la variable file_socket. Es entonces, cuando tiene todos sus parámetros definidos, el momento en el que se añade a la lista de instancias uml_switch sobre las que umlswitchd tiene información.

En caso de recibir el mensaje de que uml_switch finaliza su ejecución, se procederá a eliminar ese nodo de la lista de nodos almacenada por uml_switchd.

Cuando se reciba una petición por la tubería entre umlctl y umlswitchd, esta debe incluir el nombre de la red a la que va dirigida. Umlswitchd procesará esta petición, determinando a que uml_switch se dirige. Lo buscará en su lista, a partir del nombre que determina unívocamente cada instancia de uml_switch, y una vez encontrado, le mandará la petición correspondiente a través del socket asociado. Una vez esta comunicación haya finalizado, bien de un modo correcto, bien por timeout, se devuelve el correspondiente mensaje a través de la tubería al umlctl.

En caso de que la petición haya sido de mostrar todos las instancias que se están ejecutando, no se comunicará en ningún momento con ningún uml_switch, sino que la comunicación se reduce a umlctl y umlswitchd.

Para indicar qué comportamiento se desea en el proceso umlctl, si usar el intermediario o establecer la comunicación directa, se añade un nuevo campo que introducir al ejecutar el comando uml_switch, de carácter obligatorio, de modo que el usuario tenga que especificar bien directa para la comunicación directa, bien daemon para realizarla a través del umlswitchd.

Toda ésta lógica también necesita modificaciones en uml_switch, para procesar las peticiones, y mandar la información de que una nueva instancia comienza su ejecución, o finaliza, según el caso que corresponda. Por eso, al comenzar la ejecución se mandará el mensaje de aviso umlswitch, mientras que al finalizar se enviará el mensaje correspondiente. Hay que añadir además la lógica de espera de peticiones provenientes del umlswitchd, para lo que seguiremos el mismo método que con el umlctl, añadiremos el descriptor de ficheros al poll correspondiente, y esperando peticiones, que serán procesadas mediante funciones implementadas a tal efecto.

Se añade también la lógica asociada al procesamiento de la señal SIGTERM, en la

que se deben finalizar correctamente todas las comunicaciones, liberando los ficheros correspondientes y finalizando adecuadamente el programa. No es conveniente mandar la señal SIGKILL a este proceso para evitar posteriores complicaciones, ya que no se eliminarán los ficheros de sockets y tuberías utilizados, y en la siguiente ejecución habrá problemas de permisos.

4.3.5 Últimas modificaciones

Con todas estas modificaciones la complejidad del uml_switch ha crecido significativamente, y tiene más posibilidades de fallo, implicando ésto que puede que existan comunicaciones que no se completen. Para ayudar al usuario a determinar posibles causas de fallo, se considera interesante establecer un método de escritura de logs. Éste método ha sido utilizado en el programa uml_switch, pero su aplicación a los otros procesos es inmediata, en caso de considerarse necesaria. Mediante la función, *write_logs*, sólo hay que indicar cuál es el texto que se desea registrar, y ella escribirá la información en el fichero de logs correspondiente, junto con la hora, minuto y segundo en el que se escribe, utilizando la estructura *time_t*, para tener un mayor conocimiento del comportamiento del proceso. En principio, el fichero de logs es */tmp/logs.uml*.

Para conseguir que al lanzar el programa umlswitchd se ejecute en background, devolviendo al usuario el control de la consola, utilizaremos la técnica del doble fork de Stevens. En el código del umlswitchd queda implementado en la función *main()*, consiguiendo que el padre del proceso umlswitchd sea el proceso init, y que el proceso no finalice hasta que reciba la señal SIGKILL o SIGTERM. Para ésta última señal se ha implementado el correspondiente sig_handler.

En cuanto a las opciones de este método, se mantienen las mismas que en el caso anterior, añadiéndose además la posibilidad de listar todas las instancias de uml_switch en ejecución, con información de sus ficheros de captura, dispositivos de captura en vivo y filtro que se aplica en la captura de paquetes.

Para conseguir una total integración del nuevo uml_switch en la herramienta vnuml-parser, se realizaron algunos cambios en el código de vnuml:

- En el DTD de vnuml, se define una nueva etiqueta, capture, que es opcional, y solo puede pertenecer a otra etiqueta net. Esta etiqueta puede tener tres atributos opcionales, que son :
  - dev, para indicar el nombre de la interfaz que queremos levantar en espacio de usuario para capturar el tráfico en tiempo real
  - file, para indicar el path completo del fichero donde se quiere volcar el tráfico capturado

  - filter, para indicar el filtro de paquetes que se quiere aplicar a la captura de paquetes en el fichero. Sólo tiene sentido cuando se ha definido el atributo file.

  Además, para no interferir con el funcionamiento normal del uml_switch original, se añade un nuevo atributo a la etiqueta net, uml_switch_binary, en el que se indica el path completo del nuevo uml_switch ejecutable.

- En el módulo CheckSematincs.pm, hay dos nuevas comprobaciones:
  - Se verifica que cuando se indica el atributo filter, se ha definido al atributo file; en caso contrario se aborta la ejecución mostrando un error por pantalla.
  - Se verifica que el binario uml_switch indicado en el atributo uml_switch_binary, si está definido, es ejecutable por el usuario que ha lanzado vnumlparser; en caso contrario, aborta la ejecución mostrando el correspondiente error por pantalla.
- Los cambios pertinentes en vnumlparser.pl para procesar los nuevos atributos y etiquetas. Además, usando el módulo Net-Pcap-0.11, se verificará que la expresión introducida como filtro tiene una sintaxis válida, abortando la ejecución en caso contrario, sin llegar a lanzar los uml_switch.

Para facilitar el proceso de instalación, se adjunta un Makefile, basado en el distribuido por el proyecto original, pero modificado para añadir los nuevos programas, umlctl y umlswitchd, de modo que el usuario tan solo ha de compilarlo (make) e instalarlo, en caso de que así lo estime conveniente (make install).

4.4 Conclusiones

Como ya se ha explicado, esta versión de uml_switch se ha modificado para una utilización con vnumlparser. Pero para una mejor comprensión de las nuevas funcionalidades de uml_switch, se ofrece a continuación un resumen de todas las nuevas opciones disponibles, y su funcionamiento, por si se quiere utilizar este nuevo uml_switch independientemente:

- *uml_switch [-unix control-socket] [-hub] [-tap tap-device] [-f capture] [-expression exp] [-dev capture_dev] [-umlctl_file file] [-simulation_name sim_name net]* or
- *uml_switch -compat-v0 [-unix control-socket data-socket] [-f capture] [-expression exp] [-dev capture_dev] [-umlctl_file file] [-simulation_name sim_name net] [-hub] [-tap tap-device]*

Donde las nuevas posibilidades son :

- "-f capture" para establecer el path completo del fichero donde se escribirán los paquetes capturados. Este fichero no debería existir, o si existe, se deben tener permisos de escritura sobre el mismo. Este fichero sería sobreescrito perdiendo la información que anteriormente contuviera.
- "-expression exp", sólo puede ser usado con la opción "-f capture". "exp" es el filtro que será aplicado. Debe ir entre comillas dobles. Estas expresiones usan la sintaxis del programa tcpdump.
- "-dev capture_dev", define el nombre de la interfaz que será usada para capturar los paquetes en vivo, con la ayuda de herramientas como ethereal, por ejemplo. Esta opción sólo puede ser usada por root.
- "-simulation_name sim_name net" establece el nombre de la simulación y la red que utilizarán esta instancia de uml_switch. Es una opción necesaria si se quiere poder enviar mensajes al uml_switch en ejecución.
- "-umlctl_file", por si se quiere utilizar otra ruta distinta de la estandar, en el directorio /tmp, para establecer la comunicación con umlctl.

En cuanto al nuevo comando umlctl, a continuación se muestran las opciones correspondientes. Se recuerda que este comando ha sido diseñado para ser usado de dos modos distintos:

- Mandar instrucciones directamente al binario uml_switch
- Mandar instrucciones al binario uml_switch a través del demonio umlswitchd, lo cual ofrece más posibilidades.

Así pues, para mandar directamente ordenes a uml_switch, la sintaxis es :

- *umlctl direct command -sim_ name simulation_ name net*

donde simulation_name y net definen univocamente la simulación y red donde se quiere capturar el tráfico.

Los posibles comandos son:

- set-capture_file /ruta/al/fichero/donde/capturar, establece el fichero de captura. Es importante destacar que no se empieza a capturar.
- set-capture_dev nombre_interfaz, establece el nombre del interfaz donde capturar. En este caso, tampoco se empieza a capturar

- start-capture_file, comienza a capturar los paquetes en fichero.
- stop-capture_file, finaliza la captura de paquetes en fichero.
- start-capture_dev, comienza la captura de paquetes en la interfaz.
- stop-capture_dev, finaliza la captura de paquetes en la interfaz.
- set-capture_filter, establece la expresión filtro que será aplicado en la captura de paquetes a fichero.
- get-capture_file, devuelve la ruta completa del fichero donde se capturan los paquetes, si está definido.
- get-capture_dev, devuelve el nombre de la interfaz donde se captura los paquetes, si está definido.
- get-capture_filter, devuelve la expresión del filtro que está siendo aplicada a la captura de paquetes.

En la otra posibilidad, se utiliza como intermediaria de la comunicación el demonio umlswitchd. Para usar esta opción, umlswitchd ha de estar en ejecución cuando se lanza uml_switch, y umlctl ha de invocarse de la siguiente manera :

- *umlctl daemon command -sim_ name sim_ name net*

Los comandos son exactamente los mismos que en el caso anterior, pero ahora existe una nueva posibilidad,

- *umlctl daemon show*

que listará todos las redes y simulaciones ejecutándose en el equipo que han sido iniciadas con la opción *-simulation_ name.*

En caso de no estar ejecutándose umlswitchd, umlctl puede comunicarse con cada instancia de uml_switch, y solicitar la misma información.

Capítulo 5

Interfaz Gráfica de usuario

5.1 Introducción

Como ya se ha explicado anteriormente, VNUML para realizar simulaciones parte de un fichero XML que lee y procesa, en función de un DTD perfectamente definido. Esto, si bien ofrece enormes posibilidades de configuración y de opciones, es cierto que hace que usuarios noveles no tengan ningún tipo de facilidad de uso de la herramienta.

Es ahí donde radica la importancia de la existencia de una interfaz gráfica de usuario, que mediante una interfaz visual mucho más fácil de cumplimentar será la encargada de transcribir el fichero XML que será procesado por vnumlparser.pl. De este modo, se ofrece al usuario la abstracción de tener que crear él mismo el fichero XML.

Coincidiendo con el comienzo de este proyecto, Michel Blanc anunció la existencia de un proyecto en Sourceforge, vnumlgui, el cual perseguía justo los mismos objetivos, por lo que se nos presentó como opción idónea. Michel sentó las bases de una interfaz con casi completa compatibilidad con la versión 1.5 de VNUML. Pero el progreso de VNUML no se paró en ningún momento, publicándose rápidamente la versión 1.6 del mismo, en la que se corregían diversos errores, y se ofrecían nuevas funcionalidades. Así pues, se decidió que era fundamental portar el proyecto vnumlgui a nuevas versiones de VNUML.

Dado que en el momento de retomar esta tarea VNUML se encontraba desarrollando las primeras versiones de pruebas de la versión 1.7, fue a esta versión a la que se decidió portar la interfaz gráfica. Como ya se ha comentado, la versión original de vnumlgui fue diseñada para la versión 1.5 de VNUML.

Los cambios desde entonces han sido muy numerosos, consiguiendo una herramienta de simulación mucho más eficiente, de mejor respuesta y con más posibilidades. Debido a que la versión 1.7 sólo presenta mejoras respecto de sus antecesoras, se decidió que la mejor opción era dotar de total compatibilidad con esta versión, y dejar de soportar versiones antiguas, lo cual complicaba enormemente la tarea.

Siguiendo con la filosofía de VNUML, vnumlgui también estaba basado en software libre, en este caso utilizando Perl y las librerías GTK, lo cual facilitaba también continuar el desarrollo del mismo.

Comenzaremos esta sección con una introducción a las librerías GTK, y su adaptación al lenguaje Perl mediante los módulos de Perl-GTK, para posteriormente dedicar una sección a la implementación, donde se explicará las mejoras introducidas, y se indicarán los pasos a seguir para añadir nuevas funcionalidades.

5.2 Las librerías GTK

GTK (GIMP Toolkit) [40] es una librería para crear interfaces gráficas de usuario. Utiliza la licencia LGPG, por lo que es posible desarrollar utilizándolas sin tener que pagar ninguna cantidad por licencias. Su nombre es debido a que fueron escritas originalmente para el programa GNU Image Manipulation Program (GIMP) [42], pero actualmente son usadas por multitud de proyectos, entre los que destaca el proyecto GNOME (GNU Network Object Model Environment)[41], uno de los entornos gráficos más utilizados en sistemas Linux.

Estas librerías están compuestas principalmente de los siguientes componentes:

- GDK (Gimp Drawing Kit), encargado de todo lo relativo a dibujar (polígonos, líneas), carga y manipulación de imágenes y en general cualquier tipo de elemento gráfico. Implementan funciones de bajo nivel utilizadas para acceder a las funciones propias del sistema X.

- Glib, conjunto de bibliotecas que componen la base de Gtk y Gnome. Se encarga principalmente de todo el manejo de estructuras de datos, portabilidad, el manejo del sistema de objetos, hilos(threads) y en general todo lo que corresponde al sistema base de Gtk.

- ATK (Accesibility Toolkit), conjunto de bibliotecas que permiten que Gtk mantenga todas esas opciones de accesibilidad desde teclas especiales y dispositivos especiales.

- Pango, encargado de gestionar todo lo relacionado con el manejo de fuentes, internacionalización y afines.

A pesar de estar escritas en el lenguaje C, esencialmente es una API orientada a objetos. GTK está implementada usando la idea de clases y callbacks (punteros a funciones).

Otra de sus características fundamentales es que está basado en un modelo gobernado por eventos. Es decir, Gtk no funciona de manera lineal, no se puede anticipar cuál será la acción que el usuario elegirá para interactuar con la aplicación y es por tanto que el modelo de eventos o señales da el control a todos los sucesos debidos a la interacción con el usuario.

Este modelo está basado en un método principal, el cual casi todo el tiempo está en espera de eventos(<Gtk2-main>). Cuando ocurre un evento es añadido en una cola, y cuando le corresponda, emitirá una señal de suceso y el usuario podrá realizar la función deseada.

Es importante distinguir en GTK entre contenedores y widgets. Un contenedor se puede entender como un "recipiente" de widgets, y que como grupo, podemos establecer algunos atributos "globales" a todos ellos dentro del contenedor. Entre los contenedores, mencionar las ventanas (Gtk2::Window), los frames (Gtk2::Frame), las cajas (Gtk2::Box), los menús (Gtk2::Menubar). Todos estos heredan la mayoría de sus atributos y características como contenedores de la clase Gtk2::Container.

Por el contrario, un widget, cuyo significado literal en español es "cosa", es un componente de la GUI que ayuda a crear otros componentes, ensamblarse con otros o simplemente hacerlos funcionar. Algunos ejemplos pueden ser los botones, tablas, ventanas, listas, espacios de textos...

En general cada widget tiene su comportamiento, sus métodos y atributos. Sin embargo, podemos describir de manera general los pasos a seguir para utilizar cualquier widget:

- Instanciar el widget por medio de su constructor => Gtk2::*->new(...);
- Establecerle atributos propios o heredados.
- Conectar los manejadores hacia las señales emitidas por los eventos que nos interesen de dicho widget.
- Agruparlo en algún contenedor y darle algún atributo especial de posicionamiento.
- Mostrar el widget => Gtk2::Widget->show(...).

Es también importante distinguir entre eventos y señales. Cuando por ejemplo, un usuario pulsa con el ratón sobre un botón, arrastra un elemento de una lista, o selecciona un elemento de un menú, se corresponde a un evento. Por lo tanto, podríamos entender que un evento es la interacción entre el usuario y la aplicación.

Una vez ejecutado y conocido el evento, la manera en la se nos permite actuar en función del mismo es emitiendo un "mensaje" de regreso al programa diciendo que dicho

evento sucedió, y así el usuario pueda, mediante un callback, manipularlo. A ese mensaje se le conoce como señal.

Es importante mencionar que cuando aquí nos referimos a señal, no hablamos propiamente de señales como lo realiza un sistema operativo POSIX, que si bien puede ser una buena analogía, en realidad el manejo y manipulación de ellas se realiza de una manera muy diferente.

Para que nuestro programa realice una acción determinada tras la ejecución de dicho evento, es necesario asociar esa señal emitida con nuestra función (callback) de la siguiente manera:

- *Glib::Object->signal_ connect(objeto, señal, función, [parámetros])*

O de un modo más práctico

- *$widget->signal_ connect(evento => & funcion);*

En cuanto al posicionamiento relativo de widgets en Gtk, tiene que ver con los contenedores, tales como ventanas o cajas. Éstos tienen la habilidad de agrupar widgets y establecerles atributos de posicionamiento. Además dichos contenedores nos dan la oportunidad de un posicionamiento automático si se desea.

Por último, incluimos a continuación un resumen de los principales tipos de widgets, que utilizaremos en el desarrollo de la interfaz:

- GtkLabel : Muestra etiquetas.
- GtkTable : Empaqueta otros widgets en tablas.
- GtkEntry : Para introducir una línea de texto.
- GtkHbox : Almacena widgets en una sola fila.
- GtkVbox : Almacena widgets en una sola columna.
- GtkComboBox : Permite elegir entre una serie de opciones predeterminadas.

Como todos estos tipos de widgets heredan del mismo padre, tienen métodos comunes, entre los que destacamos, por su posterior uso:

- set_sensitive(boolean) para establecer si el usuario podrá o no interactuar con el widget. Se complementa con el método get_sensitive, para averiguar el estado del widget.
- set_text(texto), para mostrar el texto en el widget, complementado con el método get_texto.

5.3 Estudio de Vnumlgui

Una vez asentados los mínimos conceptos para poder entender el código implementado en vnumlgui, procedemos al estudio del código del proyecto, disponible en la página web *http://vnumlgui.sf.net* en su última versión estable, la 0.7. Para la aplicación de las librerías Gtk en el programa, se hace uso de GTK-Perl [43], que es un módulo que permite utilizar las librerías GTK desde un programa Perl. Utiliza la licencia GPL, por lo que tiene las mismas restricciones de uso que Perl. Se considera importante añadir esta sección porque facilita enormemente nuevas modificaciones que se decida añadir en función de nuevas funcionalidades ofrecidas por vnumlparser en futuras versiones.

Lo primero que se aprecia es que el código por completo se encuentra en el fichero *vnumlgui* del directorio src. En este fichero se aprecian los siguientes paquetes :

- El script Perl principal de la herramienta, *vnumlgui*
- El paquete *Gear*, que proporciona funciones que serán usadas en todos los paquetes, como la posición en la pantalla con sus coordenadas x e y,y demás gestiones propias de la interfaz gráfica.
- El paquete *Router*, que representa cada máquina virtual presente en la simulación. Sus atributos principales son:
  - Kernel, que indica cuál es el kernel a cargar por la máquina virtual.
  - Memory, especifica la cantidad de memoria física destinada a la máquina.
  - Filesystem y filesystem_type, para indicar el sistema de ficheros a utilizar, y si será utilizado del modo clásico o COW (Copy On Write).
  - Interfaces, uno para cada subred.
  - Execs, para los comandos a ejecutar en esta máquina virtual.
  - Xterm, que indica con qué programa lanzar los terminales asociados a las máquinas virtuales.
- El paquete *Switch*, que representa cada instancia de uml_switch que se invoca. Sus atributos principales son:
  - Mode, pudiendo ser virtual_bridge o uml_switch.
  - Type, especifica si será una subred LAN o PPP.
  - Vlan, indica si la subred es una Vlan.
  - Bandwidth, indica el ancho de banda de la red, en caso de ser PPP.

- El paquete *Host*, para representar el host físico donde se ejecutará la simulación, y que se representa con los parámetros:
  - Physical_interfaces, para las interfaces físicas del equipo.
  - Routes, para las rutas.
  - Ip_forward, por si se quiere habilitar el reenvío de paquetes, útil si se quiere proporcionar acceso a Internet desde las máquinas simuladas.
  - Execs, para los comandos a ejecutar.
- El paquete *MetaTerminal*, para la gestión de los terminales (bien con *xterm*, bien con *gnome-terminal* que se lanzarán por cada máquina virtual.

Así pues, se entiende que vnumlgui seguirá un modelo orientado a objetos, en el que cada elemento de la simulación es representado como una clase. Además, sigue el modelo de máquinas de estado, en el que se indica los diferentes estados posibles, y cuáles son los estados que pueden seguir a cada uno de ellos. Los estados en los que se puede encontrar la simulación son:

- Void, estado inicial, no hay representado ningún escenario en la herramienta.
- Clean, simulación que se ha salvado y no se ha modificado desde entonces.
- Dirty, simulación con modificaciones desde la última vez que se salvó.
- Building, simulación en espera de que vnumlparser construya la topología.
- Running, simulación cuya topología ya ha sido construida por vnumlparser.
- Execing, simulación ejecutando los comandos que se han especificado.
- Releasing, simulación que el usuario ha mandado finalizar.

El definir los estados en los que es posible se encuentre la simulación es muy útil para la interfaz gráfica. Por ejemplo, permite que si no se encuentra en el estado de clean se avise al solicitar construir la topología, o en caso de estar en el estado de running, se dibuje cada elemento de la simulación con otro color al utilizado en el resto de los estados, para que el usuario sepa a primera vista si las máquinas virtuales están en ejecución o no.

Un primer estudio de la aplicación revela que los pasos que siguen son los propios de cualquier aplicación GTK, esto es:

- Se definen los módulos de Perl a utilizar, entre los que se encuentran, claro está, los relativos a GTK.

- Se definen e inicializan las variables globales a utilizar en el programa.

- Se inicializan los contenedores y widgets, asociando los eventos a las subrutinas o callbacks. Especialmente importante en este punto es la instrucción
  Gnome2::Program->init ($CONFIG { 'NAME' } , $CONFIG { 'VERSION' });
  que se encarga de inicializar además Gtk2, necesario para cualquier aplicación que use las librerías GTK. Otra instrucción fundamental es
  $STORE { 'GLADEXML' } = Gtk2::GladeXML->
  new($CONFIG { 'GLADE_FILE' });
  donde se procesa el XML generado por GLADE para diseñar el interfaz.

- Y por último, se pasa a un bucle de espera para los eventos correspondientes, mediante la instrucción *Gtk2->main*. Este bucle no finalizará mientras el programa se ejecute.

Así pues, el proceso a seguir será diseñar mediante GLADE los widgets y contenedores necesarios, y se les asociará una subrutina al evento correspondiente. Esta subrutina será implementada en vnumlgui con las funcionalidades deseadas.

Para conseguir que las nuevas opciones introducidas, como la etiqueta capture con sus correspondientes atributos, se vean reflejadas en la interfaz gráfica, hubo que modificar el código del programa vnumlgui. Se mostrará más adelante el proceso seguido para añadir las etiquetas y atributos relativos a la captura de tráfico en la interfaz.

Continuando con el estudio del código presente en el script, nos centramos a continuación en el proceso de la clase principal de vnumlgui, pues el resto son utilizados para representar instancias, y su función es más sencilla de comprender. De este modo, se aprecia que, después de establecer las variables globales de configuración que serán utilizadas, se inicializan aspectos como:

- Diversas variables, como el estado de la simulación a vacío, o la escala con la que se dibujarán las instancias. Es de especial importancia señalar que aquí se carga el fichero XML donde se lee la configuración generada por el programa Glade.

- La configuración, que se cargará del directorio personal del usuario, en el fichero oculto *.vnumlgui*. En caso de no existir, se cargará la configuración por defecto tal y como la establece vnumlgui.

- Variables relativas a la representación de la simulación, como el tamaño de la misma, si utilizar scroll, o aliases.

- Los parámetros de las clases relativas a la representación de los iconos.
- Las listas de los kernels y sistemas de ficheros que se mostrarán por defecto cuando el usuario lo solicite, o las listas de claves públicas ssh disponibles.

Posteriormente, se indica que todos los eventos que se produzcan los gestionará este paquete, y se comprueba si al invocar vnumlgui se le ha pasado un fichero XML que contiene el escenario a simular. En caso afirmativo se procede a abrirlo, y en caso contrario se establece el estado a vacío. En cualquier caso, a continuación se comienza con el bucle principal que estará a la espera de los eventos que se produzcan con la interacción del usuario. El *exit* que aparece después no se llegará a ejecutar, pero se añade por mayor claridad en el programa.

Se ha comentado anteriormente que uno de los aspectos que se inicializa es cargar el fichero XML generado por Glade. Esta es una herramienta que facilita enormemente el desarrollo de los interfaces GTK, generando un fichero XML que es interpretado por el módulo Perl GladeXML.pm. Para ello, usaremos la herramienta glade-2 con la librería libglade. Este fichero de configuración es *share/vnumlgui.glade*.

Es en este fichero donde se determina la apariencia que tendrá la interfaz gráfica, con sus botones y opciones. De este modo, se diseñará un widget para cada cuadro de diálogo que se desee incluir en la interfaz. Se podrá definir por cada widget una serie de propiedades, consistentes en:

- Propiedades propias del widget, como en nombre, tipo de widget, anchura, altura, posición, si incluirá bordes...
- Empaquetamiento del widget, en la que se define, entre otras cosas, la posición relativa que ocupará este widget en el contenedor al que pertenece.
- Propiedades en las que se definirá si por defecto el widget es visible y si el usuario podrá interaccionar con él.
- Por último, se definirá qué tipo de señales se asociarán, y cual es el gestor encargado de realizar las operaciones necesarias en caso de recibir la señal. Este gestor será una función implementada en vnumlgui.

Así, pues, el proceso de añadir nuevas funcionalidades se puede resumir con los siguientes pasos:

- Se definirá mediante Glade en que widget se incluirá nuestra nueva funcionalidad (si pertenece al cuadro de diálogo de switch, de router, de host, de abrir una simulación, de propiedades generales...)

- Se determinará qué tipo de widget es, si se utilizará para mostrar un nombre, para seleccionar una entre un conjunto de opciones disponibles, para introducir un valor...
- Además, se definirá si será sensible a interacciones con el usuario por defecto.
- Por último, dentro de Glade, se le asociará qué función gestionará las señales recibidas.
- Se implementará esta función en vnumlgui, con sus operaciones asociadas.

Muestra de ello, es por ejemplo el cuadro principal, en el que se aprecia en Glade que cada opción tiene su método asociado, por ejemplo:

- Opción nuevo del menú File, asociado al método *on_ mainMenuFileNew*.
- Opción open del menú File, asociado al método *on_ mainMenuFileOpen*.
- Opción copy del menú Edit, asociado al método *on_ mainMenuEditCopy*.
- Opción cut del menú Edit, asociado al método *on_ mainMenuFileCut*.

Todos estos métodos están implementados tras el método main ya comentado. Por ejemplo, podemos observar cómo se ha implementado el método on_mainMenuFileNew:

```
sub on_mainMenuFileNew() {
  my ($widget, $event) = @_;
  &debug("$widget received $event");

  return &create_new();
}
```

en el que se destaca :

- Con la primera línea, se recibe tanto el widget que ha enviado el evento asociado al método, como el propio evento.
- Mediante el método *debug* se mostrará por pantalla esta información, en caso de que vnumlgui se esté ejecutando con esta opción, muy útil para estudiar posibles casos de fallo.
- Por último, se procede al propio método de crear una nueva simulación.

Una vez entendido el procesamiento, se explica a continuación las diferentes modificaciones implementadas en vnumlgui, para posteriormente explicar los pasos seguidos para añadir en el programa la opción de definir en la interfaz misma las nuevas opciones de captura de tráfico. Esto se espera sirva para explicar cómo añadir futuras funcionalidades que aparecerán en nuevas versiones de vnumlgui.

5.4 Modificaciones implementadas

Con el objetivo de comenzar a familiarizarse con el proyecto, el primer punto en el que nos centraremos será la modificación de los puntos necesarios para conseguir una compatibilidad básica con vnumlparser versión 1.7. Posteriormente nos centraremos en añadir las nuevas funcionalidades.

Puesto que los cambios entre las versiones 1.6 y 1.7 en el DTD definido de vnuml no son tan numerosos, ya que en principio sólo se añaden nuevas etiquetas y atributos opcionales, nos centraremos en los cambios producidos entre las versiones 1.5 y 1.6, causantes de la no compatibilidad de las herramientas:

- Nuevas etiquetas: *<ssh_ version>*, *<vm_ mgmt>*, *<mgmt_ net>*, *<user>*, *<group>*.
- Etiquetas eliminadas : *<ip_ offset>* , puesto que se usan en su lugar los atributos *network, mask* y *offset* de la etiqueta *<vm_ mgmt>*.
- El atributo version de *<ssh_ key>* ha sido sustituido por *<ssh_ version>*.
- Hasta la version 1.5 el atributo *type* de la etiqueta *<vm_ mgmt>* era por defecto *private*.
- Anteriormente, *<host_ mapping>* era hijo de *<global>*, ahora lo es de *<vm_ mgmt>*.
- Las etiquetas *<default_ kernel>* y *<kernel>* tienen un atributo *initrd* opcional, que especifica la ruta completa del fichero initrd para los kernels que lo soportan.

En principio, el problema con estas etiquetas y atributos era el cambio del DTD que define como debe estar escrito el XML. Por lo que para la correcta generación del XML hemos de modificar el método dump_xml. Este método está particularizado para cada instancia de los tipos host, router y uml_switch.

Una de las peticiones existentes en cuanto al comportamiento de vnumlgui era añadir la posibilidad de guardar la posición relativa de los elementos que forman la simulación. Para añadir esta funcionalidad se utilizaron las directivas de procesamiento. De este modo, cada vez que un usuario solicite abrir el XML correspondiente a la simulación con

la que estuvo trabajando, los elementos de la misma estarán situados tal y como él los ordenó.

Las instrucciones de procesamiento se utilizan para transmitir información a una aplicación. Permiten de este modo que un documento XML contenga instrucciones para una aplicación, y no son objetivo de un parseador XML. Deben empezar con <? y finalizar con ?>. Por estas características, parece una alternativa idónea para pasar información a la aplicación vnumlgui sin modificar el procedimiento del parser de vnuml.

Así pues, cuando un usuario solicita guardar una simulación, se añadirá en las etiquetas host, net y vm la información relativa a su posición en coordenadas cartesianas, x e y. Y para que estas directivas no sean procesadas por el parser de vnumlgui, serán directivas de preprocesamiento. El añadir la funcionalidad de guardar esta información en el xml salvado es trivial, mientras que para que vnumlgui sí tenga en cuenta estos valores, se implementa utilizando la función getData proporcionada por el módulo Perl DOM de XML.

Se comprobó además que cuando desde la interfaz gráfica se solicitaba lanzar la simulación, si ocurría algún error, el programa vnumlgui se quedaba en un estado indeterminado de error del que sólo se podía salir reiniciando la aplicación. Esto era debido al problema conocido de vnumlgui con los entornos X, por el cual, a veces no se lanzan las xterm correspondientes a la máquina virtual que se desea simular. Vnumlgui no estaba preparada para esta situación, y se decidió añadir un timeout en vnumlgui para la espera en lanzar cada máquina virtual. De este modo, se esperaría los mismos segundos indicados en el timeout de vnumlparser a que se lanzara esa simulación. Para conseguir esto, hubo que añadir a vnumlparser la orden de que en caso de que la máquina se lanzase correctamente, se mostrase por pantalla un mensaje de información. Éste mensaje es el que espera vnumlgui para considerar que la máquina ha sido correctamente lanzada. Se intentó mediante otros métodos, como recogiendo el mensaje $MCONSOLE\_USER\_NOTIFY$ de uml_switch que espera vnuml, pero se decidió que ésta era la mejor solución.

Otro de los objetivos que se planteó era unificar el proceso de parseo del XML entre ambos procesos, pero ésto se ve enormemente complicado debido a la diferente estructura de ambos procesos, ya que vnumlparser es claramente lineal, mientras que vnumlgui debe seguir una aproximación mucho más orientada a objetos, siguiendo el sistema de eventos propio de GTK.

Pero sí que se consideró interesante unificar el proceso de parseo y generación del XML en la medida de lo posible. Para ello, se decidió utilizar los métodos y funciones implementados en los módulos desarrollados por el equipo de vnuml. De este modo, nos aseguramos que en futuras versiones estos métodos seguirán funcionado en vnumlgui. Así pues, en la funcion open_file de vnumlgui se utilizará el módulo DataHandler, y las

funciones *get_ version*, *get_ vmmgmt_ type*, *get_ default_ filesystem*...

Además, para mejorar la comunicación entre los procesos vnumlgui y vnumlparser, se decidió utilizar el módulo Open2 de IPC. De este modo es posible ejecutar la función

- *$pid = open2($salida, $entrada , $comando);*

donde se ejecuta el "comando" pudiendo leer los mensajes que muestra por pantalla en el descriptor "$salida" e introducirle datos por el descriptor "$entrada". De este modo, podemos leer la salida del comando "vnumlparser" de un modo más sencillo de como se hacía hasta ahora. Esto se ejecutará cuando se mande crear una simulación, ejecutar comandos en la misma o finalizarla.

5.5 Añadiendo nuevas funcionalidades : etiquetas de captura en uml_switch

En esta sección se detallará más en profundidad cómo se han añadido las etiquetas relativas a la captura en el uml_switch, que son:

- Atributo *dev* de la etiqueta *capture*.
- Atributo *file* de la etiqueta *capture*.
- Atributo *filter* de la etiqueta *capture*.
- Atributo *uml_ switch_ binary* de la etiqueta *uml_ switch*.

En primer lugar, definimos cuáles son los objetivos y posibilidades:

- Se añadirá en la ventana correspondiente a los atributos relativos a la red dos campos para cada atributo, uno de texto, como "capture file", y otro una caja donde introducir el valor correspondiente.
- En estas cajas de texto sólo se podrá introducir algún valor cuando tenga sentido con el resto de los valores establecidos por el usuario. Por ejemplo, si se define que la red sea del tipo "hub" no se podrá escribir.
- En el momento en que el usuario introduzca algún valor en alguna caja, ha de reflejarse en el XML. Además, cuando el usuario vuelva a abrir la ventana, verá el valor anteriormente introducido.
- El comportamiento descrito en el punto anterior ha de ser el mismo si el usuario escoge la opción de abrir un fichero XML.

Lo primero que hemos de hacer es diseñar la apariencia en Glade. Para la visualización y comprensión de los widgets y containers, es muy recomendable activar la posibilidad de ver el árbol de widgets, como se refleja en el cuadro 5.1. Consideramos que la etiqueta capture es un detalle que debería pasar transparentemente al usuario, por lo que se añadirán todos los widgets relativos en el mismo cuadro de diálogo de uml_switch, situados en el widget *switchDialog*. Este widget contiene otro widget, de la clase *GtkVBox*, formado por un cuadro de diálogo de botones, con posibilidades de *Ok* y *Cancel*, y una tabla, que contiene, entre otros elementos:

- Un widget de la clase *GtkLabel* para la etiqueta que indica el campo del nombre, con un campo para introducirlo mediante un widget *GtkEntry*.
- Una etiqueta, mediante otro widget *GtkLabel*, para indicar el campo del modo de uml_switch, y un widget *GtkComboBox* para elegir la opción del modo (entre las predeterminadas, virtual_bridge y uml_switch).
- Un widget de una línea *GtkHbox* que contiene a su vez dos widgets, uno para marcar si el uml_switch formará una VLAN, y otro para indicar su identificador.
- Otro widget para seleccionar la interfaz externa, en caso de querer usarla. Se seleccionará con un widget de la clase *GtkComboBox* para evitar que el usuario pueda introducir algún valor no válido.
- Mediante un widget de la clase *GtkVbox*, se añadirán dos widgets de tipo *GtkRadioButton* para elegir entre LAN o PPP.

Así pues, lo lógico sería aumentar el número de filas de la tabla, para poder introducir nuestros nuevos campos. Para ello, seleccionamos la tabla *switchTable*, y en el campo *rows*, lo aumentamos en 4, uno para cada opción nueva de captura que queremos añadir. Se añadieron los siguientes widgets:

- sdEntSwCapDev, caja de texto donde introducir el atributo dev de la etiqueta capture, junto con el widget de la clase GtkLabel sdLblSwCapDev.
- sdEntSwCapFile, caja de texto donde introducir el atributo file de la etiqueta capture, junto con el widget de la clase GtkLabel sdLblSwCapFile.
- sdEntSwCapExp, caja de texto donde introducir el atributo exp de la etiqueta capture, junto con el widget de la clase GtkLabel sdLblSwCapExp.
- sdEntSwBin, caja de texto donde introducir el atributo uml_switch_binary de la etiqueta net, junto con el widget de la clase GtkLabel sdLblSwBin.

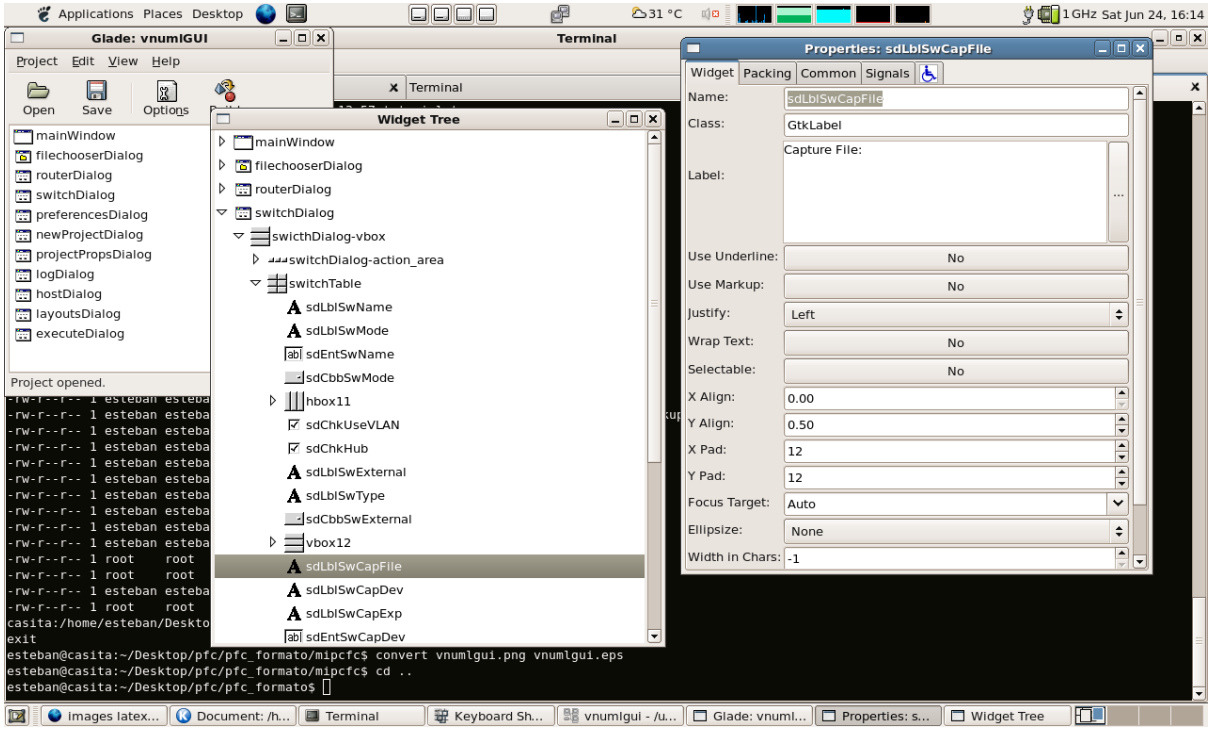

Figura 5.1: Interfaz Glade

Mostraremos el proceso seguido para añadir la opción de volcar a fichero la captura de paquetes, en cualquiera de las otras tres opciones el proceso es exactamente igual.

Definiremos un widget de clase *GtkLabel*, de nombre *sdLblSwCapFile*, que mostrará por pantalla *Capture File:*, y así se le indicará en la opción *Label*. Se definirá en la opción Packing, que ocupará en el contenedor la posición 0 en el eje x. Se definirá otro widget de clase *GtkEntry* de nombre *sdEntSwCapFile*, cuya posición en el eje x será 1, misma posición en el eje y que el widget anterior. Se marcarán como afirmativo las opciones *Editable* y *Text Visible* dentro de la opción *Widget*. Dentro de la opción *Common*, especificaremos a sí las opciones *Visible* y *Sensitive*.

De este modo, se podrá introducir valores por defecto en la etiqueta de captura de paquetes en un fichero. Una vez finalizado este proceso, se pueden ver los resultados, indicando a vnumlgui que cargue este nuevo fichero. Para ello, basta con sobreescribir el fichero
/usr/local/share/vnumlgui/vnumlgui.glade, siendo recomendable realizar una copia de seguridad del fichero, para evitar posteriores problemas. Ahora, se puede comprobar si el widget se muestra como deseamos, pero vnumlgui no lo puede interpretar aún. Para esto, hemos de modificar este script como se explica a continuación.

Todas estas modificaciones aparecerán en el cuadro de diálogo *Net properties*. Para no complicar la tarea al usuario, los widgets de entrada de datos sólo permitirán introducir

texto para definir, por ejemplo, el fichero de captura, en caso de que el modo de la red sea sólo uml_switch. Para ello, en Glade vemos que el modo de uml_switch se define con el método *on_ sdCbbSwMode_ changed*. Hemos de añadir en vnumlgui una condición que compruebe el modo de la red, y sólo en caso de ser uml_switch, definir como sensible todos los widgets de entrada. Para ello, introduciremos el siguiente fragmento de código :

```
   sub on_sdCbbSwMode_changed() {
     my ($widget, $event) = @_;
     &debug("$widget received $event");

     if ($STORE{'GLADEXML'}->get_widget('sdCbbSwMode')->
get_model->get($STORE{'GLADEXML'}->get_widget('sdCbbSwMode')
->get_active_iter) eq 'uml_switch') {
       $STORE{'GLADEXML'}->get_widget('sdChkHub')
->set_sensitive(TRUE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapExp')
->set_sensitive(TRUE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapDev')
->set_sensitive(TRUE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapFile')
->set_sensitive(TRUE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwBin')
->set_sensitive(TRUE);
     } else {
       $STORE{'GLADEXML'}->get_widget('sdChkHub')
->set_sensitive(FALSE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapFile')
->set_sensitive(FALSE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapDev')
->set_sensitive(FALSE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwCapExp')
->set_sensitive(FALSE);
       $STORE{'GLADEXML'}->get_widget('sdEntSwBin')
->set_sensitive(FALSE);

     }
```

```
}
```

Mediante Glade, se define que en caso de cambiar el modo de la red, se gestione con esta función, por lo que no tenemos que añadir nada al respecto en vnumlgui. Pero aún hay código que añadir para que el resultado sea el que queramos. Es recomendable que una vez que el usuario haya introducido algo en este campo de texto, cada vez que vuelva a esta ventana de propiedades de red, aparezca el texto que escribió. Esto es también aplicable en el caso de que el usuario haya abierto una simulación definida en un fichero XML. Todo esto se implementa en el método dialog_run() de la clase Switch, que a su vez es llamado desde el método _on_item_properties() del paquete Gear, método asociado al evento producido cuando se selecciona el widget correspondiente :

```
 if (defined $self->{'mode'} and $self->{'mode'} eq 'uml_switch') {
    $STORE{'GLADEXML'}->get_widget('sdChkHub')->set_sensitive(TRUE);

    $STORE{'GLADEXML'}->get_widget('sdEntSwCapDev')
->set_sensitive(TRUE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapExp')
->set_sensitive(TRUE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapFile')
->set_sensitive(TRUE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwBin')
->set_sensitive(TRUE);

    $STORE{'GLADEXML'}->get_widget('sdEntSwCapDev')->
set_text($self->{'capture_dev'});
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapFile')->
set_text($self->{'capture_file'});
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapExp')->
set_text($self->{'capture_exp'});
    $STORE{'GLADEXML'}->get_widget('sdEntSwBin')->
set_text($self->{'uml_switch_binary'});

    $STORE{'GLADEXML'}->get_widget('sdChkHub')->set_active(TRUE)
if defined $self->{'hub'};
  } else {
    $STORE{'GLADEXML'}->get_widget('sdChkHub')
->set_sensitive(FALSE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapDev')
```

```
->set_sensitive(FALSE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapExp')
->set_sensitive(FALSE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwCapFile')
->set_sensitive(FALSE);
    $STORE{'GLADEXML'}->get_widget('sdEntSwBin')
->set_sensitive(FALSE);
  }
```

Para ello, hemos de añadir dentro de los atributos que definen a la clase uml_switch, en el hash *switch_fields*, los relativos a capture_file, capture_dev, capture_filter y uml_switch_binary, quedando

```
my %switch_fields = (
                        'mode'       => 1,
                        'type'       => 1,
                        'external'   => 1,
                        'hub'        => 1,
                        'vlan'       => 1,
                        'bandwidth'  => 1,
                        'uml_switch_binary'  => 1,
                        'capture_file'  => 1,
                        'capture_exp'  => 1,
                        'capture_dev'  => 1,
                      );
```

En el mismo método de *dialog_run*, añadiremos lo siguiente para inicializar las variables en caso de que el usuario introduzca algún valor:

```
    if ($self->{'mode'} eq 'uml_switch'){
        $self->{'capture_file'} = $STORE{'GLADEXML'}->
get_widget('sdEntSwCapFile')->get_text();
        $self->{'uml_switch_binary'} = $STORE{'GLADEXML'}->
get_widget('sdEntSwBin')->get_text();
        $self->{'capture_exp'} = $STORE{'GLADEXML'}->
get_widget('sdEntSwCapExp')->get_text();
        $self->{'capture_dev'} = $STORE{'GLADEXML'}->
get_widget('sdEntSwCapDev')->get_text();
    }
```

En caso de que alguna de estas variables se haya inicializado con algún valor, se añadirán al XML relativo a la simulación. Para ello, en la funcion dump_xml() de la clase

Switch, invocado desde métodos como simulation_syntax_check() o save_simulation(), por ejemplo, se añade lo siguiente :

```
   if ((uc $self->{'type'} eq 'PPP') || ($self->{'capture_file'}) ||
($self->{'capture_dev'}) || ($self->{'capture_filter'})) {
       $xml .=">";
       $xml .= "\n\t\t<bw>" . $self->{'bandwidth'} . "</bw>"
if (uc $self->{'type'} eq 'PPP');
       if ($self->{'capture_dev'}){
           $xml .= "\n\t\t<capture dev=\"" . $self->{'capture_dev'} . "\"";
           $xml .= " exp=\"" . $self->{'capture_exp'} . "\""
   if ($self->{'capture_exp'});
           $xml .= " file=\"" . $self->{'capture_file'} . "\""
   if (\$self->{'capture_file'});
           $xml .= " />";
       }
       elsif ($self->{'capture_exp'}){
   $xml .= "\n\t\t<capture exp=\"" . $self->{'capture_exp'} . "\"";
           $xml .= " file=\"" . $self->{'capture_file'} . "\""
   if ($self->{'capture_file'});
           $xml .= " />";
       }
       elsif ($self->{'capture_file'}){
           $xml .= "\n\t\t<capture file=\"" . $self->{'capture_file'} . "\"";
           $xml .= " />";
       }
       $xml .= "\n\t\t<?vnumlgui =\"". $self->{'x'} ."\" y=\"".
   $self->{'y'} ."\"?>\n";
       $xml .= "\n\t</net>\n"

     }
     else {
       $xml .= ">\n\t\t<?vnumlgui =\"". $self->{'x'} ."\" y=\"".
   $self->{'y'} ."\"?>\n";
       $xml .= "\n\t</net>\n"
     }
```

Y para leer los campos correspondientes al elegir la opción de abrir el XML, en método open_file() de la clase principal se añade:

```
    if ($net->getElementsByTagName("capture")->getLength > 0) {
        $neth{'capture_file'} = $net_list->item($i)->
getElementsByTagName("capture")->item(0)->getAttribute("file");
        $neth{'capture_exp'} = $net_list->item($i)->
getElementsByTagName("capture")->item(0)->getAttribute("exp");
        $neth{'capture_dev'} = $net_list->item($i)->
getElementsByTagName("capture")->item(0)->getAttribute("dev");
    }
```

Éste método de open_file() se llama o bien al ejecutar el propio programa vnumlgui, si se le pasa algún parámetro añadido que se supone es el fichero a abrir, o desde el método open_simulation(), método a su vez llamado desde on_toolbuttonOpen_clicked, que es la función asociada al evento de pulsar sobre la opción de "open" del menú de la interfaz gráfica. La variable $neth no es más que un hash de Perl que será utilizado para introducir elementos en los campos correspondientes al hash switch_fields, que es donde se guardará toda la información relativa a cada uml_switch.

5.6 Conclusiones

Tras la realización de esta tarea, se considera que vnumlgui puede ser utilizado para la creación de escenarios que serán posteriormente simulados mediante vnumlparser. Se han realizado numerosas pruebas, intentando seleccionar la mayoría de las opciones existentes en vnuml, y el resultado ha sido satisfactorio, consiguiendo crear ficheros xml siguiendo el dtd de la incipiente versión 1.7 de vnuml. Se ha comprobado además que la herramienta es capaz de interpretar correctamente cualquiera de los ejemplos incluidos con vnuml, y su posterior generación.

Todo el código aquí comentado se encuentra públicamente disponible en la web del proyecto en sourceforge. Para acceder al mismo, se ha de descargar la última versión disponible del CVS. Para facilitar su uso, se ha modificado el fichero Makefile, y con un simple make, se instalará la aplicación.

Mediante esta tarea, he podido conocer una de las aproximaciones seguidas para la creación de interfaces gráficos, con un modelo de eventos, como es el utilizado por GTK. Sin duda, es una aproximación muy distinta a la de la programación en lenguajes más clásicos, como los utilizados en las otras tareas del proyecto. Además, me ha permitido estudiar otra aproximación en la comunicación entre procesos, como es la llevada a cabo mediante el lenguaje Perl entre vnuml y vnumlparser.

Y mediante la explicación de los pasos seguidos para añadir las nuevas funcionalidades, se espera que la ampliación de las posibilidades de vnumlgui a medida que vnumlparser aumente de versión sea mucho más sencilla, evitando de este modo que la interfaz gráfica quede obsoleta y no sea recomendable su utilización.

Como ejemplo, se muestra en la figura 5.2 una simulación construida mediante vnuml-gui.

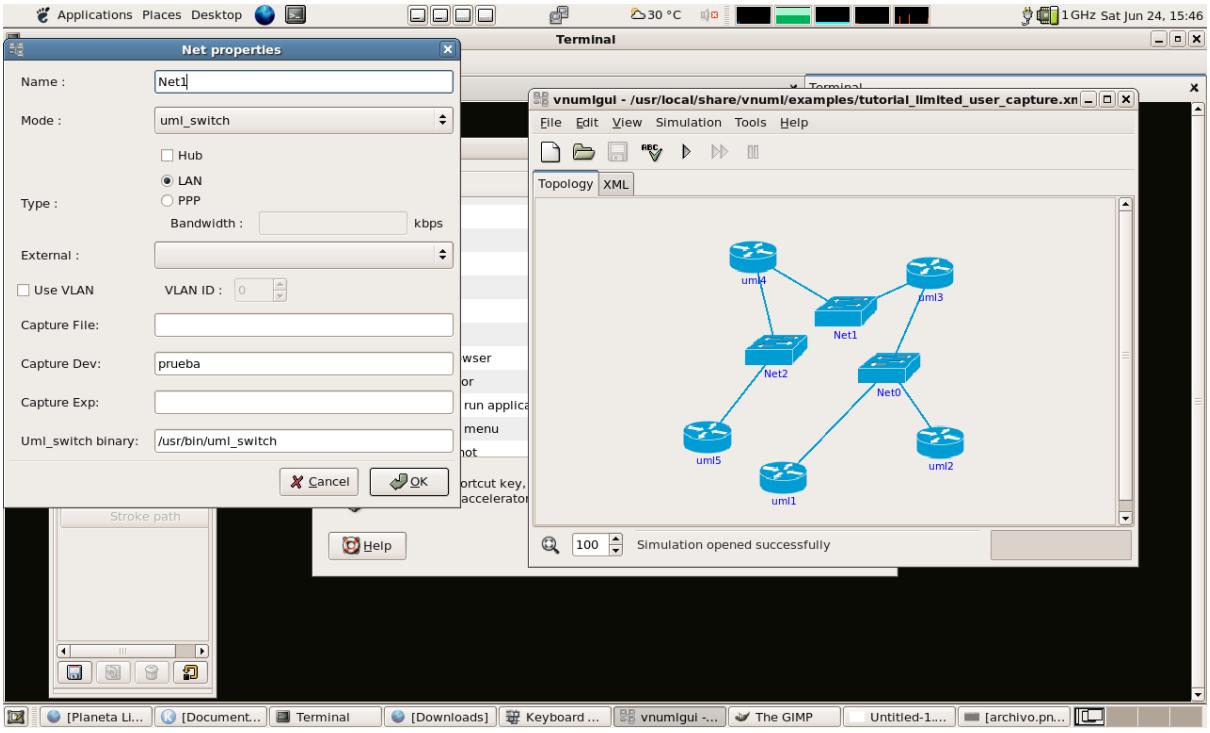

Figura 5.2: Imagen Vumlgui

Capítulo 6

Conclusiones y Futuros Trabajos

El trabajo que se describe en este proyecto ha buscado contribuir de forma significativa al proyecto VNUML, realizando tareas como facilitar las posibilidades de probar la herramienta a los posibles usuarios, añadiendo la funcionalidad del análisis del tráfico de la red y la integración con la interfaz gráfica.

Es una diferencia significativa con los proyectos en los que se comienza con un desarrollo completo, partiendo desde el inicio. Por esto, la primera parte del proyecto fue un estudio pormenorizado tanto de los procesos habituales de virtualización, como de la herramienta en sí. En este caso, no bastó con el estudio del código del proyecto VNUML, también se estudió el código fuente del software uml_router, incluido en el paquete uml_utilities, y que es el utilizado para gestionar la red simulada. Todo este trabajo se intenta reflejar en el primer capítulo del proyecto. A continuación, se procedió al desarrollo de las diferentes tareas que componen el proyecto. Analizaremos a continuación el grado de cumplimiento de los objetivos planteados en la sección correspondiente.

En primer lugar, por considerarse una de las tareas más necesarias a acometer, se encuentra el desarrollo de un CD autoarrancable que incluya la herramienta. Se ha comprobado que esta funcionalidad puede ser un factor decisivo en el uso de la herramienta, pues es fácil comprobar que el proceso de instalación puede acarrear numerosos problemas a usuarios no familiarizados con el entorno (este hecho se puede ver reflejado muy fácilmente en la lectura de los archivos de la lista de usuarios de la herramienta). Además, debido a las posibilidades que presenta, es una herramienta muy útil para usuarios avanzados en situaciones puntuales. Teniendo perfectamente definido el entorno deseado, se desarrolló una imagen de libre distribución que se puede encontrar en la página web.

Posteriormente, se acometió el estudio de incorporar un analizador de protocolos en la red. Hasta ahora, se podía capturar el tráfico relativo a un equipo mediante herramientas como tcpdump ejecutadas en los mismos. Pero se consideraba interesante, principalmente por motivos académicos, que el usuario pudiera estudiar el tráfico existente en toda la

red. Una vez implementado, se añadieron más funcionalidades tanto para facilitar su uso como para cumplimentar otros requisitos deseados. Es por ello que se desarrolló también tanto la posibilidad de definir qué tráfico era interesante capturar, como de capturarlo a medida que el tráfico se produce. Por último, se comprobó que sería de gran ayuda tener la posibilidad de cambiar el comportamiento del capturador de tráfico en tiempo de ejecución, indicando, por ejemplo, en qué momento era interesante comenzar la captura, o finalizarla, o cambiar el patrón de los paquetes a capturar... Además, esto permitiría la existencia de un comando a través del cual poder comunicarse con la red correspondiente, lo cual facilitaba bastante su utilización.

Por último, otra de las tareas a realizar, fundamental para acercar el uso de la herramienta a los usuarios, es una interfaz gráfica que hiciera transparente al usuario la tediosa necesidad de definir el escenario a simular en un fichero XML. Sin duda, en el objetivo de aumentar el número posible de usuarios de VNUML, este era un paso fundamental. De modo que cuando se presento el proyecto vnumlgui, parecía ser la alternativa idónea. Implicaba un estudio inicial del código del proyecto, a cambio de facilitar enormemente el trabajo, pues tan sólo sería necesario adaptar la versión del software soportado por la herramienta, 1.5, a la actual, 1.7. Además del proceso de adaptación de los cambios, se añadieron las nuevas funcionalidades desarrolladas por VNUML, y se corrigieron algunos fallos menores.

Como tareas menores, se intentó contribuir en la medida de lo posible con otros aspectos del proyecto, como es la actualización de los ejemplos publicados en la web que reflejan algunas de las posibilidades de escenarios simulados que permite la herramienta.

Aunque se considera que todas estas tareas se han desarrollado completa y satisfactoriamente, es cierto que cualquier proyecto de ingeniería puede ser ampliado y mejorado. En este caso, existen ciertas tareas que se considera interesante afrontar, entre las que se encuentran las siguientes:

- Imagen arrancable para dispositivos USB. Hoy en día, el uso de dispositivos de almacenamiento masivo USB está muy extendido. Actualmente, un tamaño de los mismos de 1 o 2 GB es lo habitual, en vez de los 700 Megabytes de los CDs utilizados en este proyecto. Además, suponen un paso intermedio con los DVDs en cuanto a capacidad, pues los 4 GB de éstos se antoja excesiva para las necesidades del proyecto. Además, se considera un dispositivo USB más práctico porque permite posteriores escrituras, lo cual significa que los datos de las sesiones de los usuarios pueden guardarse sin problemas. Además, se considera que la mayor parte del proceso de generación de la imagen para el CD sería análogo, salvo diferencias como el gestor de arranque a utilizar.

- Otra de las tareas que pudiera ser interesante es la automatización del proceso de creación de la imagen. En el apéndice se recogen las instrucciones necesarias para la creación de nuestro CD remasterizado, pero pudiera ser interesante la creación de scripts que automaticen el proceso. Es cierto que ya existen scripts encargados de la remasterización, pero no incluyen la funcionalidad de recompilación del kernel, sin duda por ser el proceso de mayor dificultad. Pero puesto que la idea es crear un kernel lo más genérico posible, estudiando el comportamiento de los scripts encargados de configurar el kernel podría llegar a ser posible.

- Con la creación del CD, se espera que los usuarios puedan probar la aplicación sin mayores problemas. Esto se espera disminuya las dificultades del primer paso en el uso de la herramienta, que es su divulgación y conocimiento. Pero no facilita la instalación de la herramienta para su uso habitual. Sin duda, la mejor solución para este problema es la creación de paquetes software para cada distribución, que permite la instalación de nuevo software sin mayores problemas gracias a instaladores de software. Actualmente, se están produciendo avances al respecto para la distribución de Gentoo, así que se recomendaría seguir la misma aproximación en la creación de paquetes .deb, para Debian y derivados, y .rpm, para Fedora Core Linux y derivados. De este modo, se podría instalar en la mayoría de las distribuciones utilizadas actualmente.

- Envío de mensajes a otras instancias uml_switch otros equipos. Debido al uso del uml_switch actual, se ha decidido que la comunicación entre procesos se establezca siempre en el ámbito local. Si bien es cierto que esto hoy en día no supone ningún tipo de limitación, es cierto que en un futuro sí que podría llegar a suponerlo. Se considera en este proyecto que la idea de que VNUML pudiera utilizar varios equipos para llevar a cabo una simulación ayudaría enormemente a mejorar las prestaciones y posibilidades de simulación de escenarios. En caso de que llegase a distribuirse entre varios equipos, sería interesante la capacidad de establecer la comunicación entre los procesos a través de protocolos sobre IP, estudiando la viabilidad del uso de TCP o de UDP.

- Otro aspecto que podría ser interesante llevar a cabo es la completa integración en el desarrollo tanto de la interfaz gráfica como de la herramienta en sí. Es cierto que puede que en principio, ambos proyectos no estén orientados al mismo segmento de usuarios, pero si se consiguiese unificar parte del desarrollo, es seguro que se ahorraría gran parte del trabajo, no necesitando desarrollar cada nueva funcionalidad siguiendo dos aproximaciones distintas.

- Uno de los grandes problemas que acarreaba la actualización de la versión soportada de la interfaz gráfica es la decisión de qué versiones de VNUML serían soportadas. Desde un principio, fue evidente que soportar distintas versiones complicaba en exceso la herramienta, pues el DTD de VNUML ha sufrido numerosos cambios y mejoras desde la versión 1.5. Así pues, se decidió por conseguir una interfaz completamente funcional con la versión 1.7, posponiendo el soporte de otras versiones anteriores. Esto puede que para la rama 1.X no sea crucial, pues siempre es recomendable actualizar las versiones debido a que se corrigen fallos y se añaden nuevas funcionalidades, pero si como está previsto se publica una rama 2.X paralela, sí que sería interesante soportar ambas.

Apéndice A

Remastering Knoppix Howto

A.1 Introduction

This document will try to explain how to remaster a Knoppix CD in order to get a new CD with the vnuml 1.6 software. In addition to this, it includes how to recompile the kernel with skas patch (Separate Kernel Address Space) to get a better handle of memory. This have been tested on:

- Knoppix 4.02 CD
- kernel host 2.6.13
- vnuml 1.6

All the steps described here have been made over a stable Debian with 2.6.13 kernel, although it would be possible to finalize the process successfully from the CD if we don't reboot the host in the specified moment.

A.2 Setting up the environment

First of all, we have to boot from Knoppix CD, and we will mount our hard drive partition as root user with read and write permissions. In this document, this partition will be /dev/hda1.

- *mount -rw /dev/hda1 /mnt/hda1*
- *mount -rw /dev/hda1 /mnt/hda1*
- *mkdir /mnt/hda1/knx*

We need at least 1 GB free of RAM+SWAP due to the use of cloop script, which is the compress software we recommend to use. If you don´t have enough RAM, you can follow these instructions in order to get extra 750 MB of swap:

- *cd /mnt/hda1/knx*
- *dd if=/dev/zero of=swapfile bs=1M count=750*
- *mkswap swapfile*
- *swapon swapfile*

We need to copy all software present in CD to the hard drive. We will make 2 directories, one for our new live CD, and the other for the source of CD, on the disk partition that was previously mounted. Also, we make additional directories under these named KNOPPIX:

- *mkdir -p /mnt/hda1/knx/master/KNOPPIX*
- *mkdir -p /mnt/hda1/knx/source/*
- *cp -Rp /KNOPPIX/ /mnt/hda1/knx/source/KNOPPIX*
- *cp /cdrom/index.html /mnt/hda1/knx/master/*
- *cd /cdrom*
- *find . -size -10000k -type f -exec cp -p –parents {} /mnt/hda1/knx/master/*

A.2.1 Installing and removing Debian packages

Now we have two choices, reboot the host and continue working from the operating system installed, or continue working from the CD. I reboot because the performance and speed is better, but if we don´t have installed any GNU/Linux distro we have to continue working from CD.

We execute the following command:

- *chroot /mnt/hda1/knx/source/KNOPPIX*

in order to work with the packages installed on our new CD.

The better way to install or remove program is using apt command. If you prefer don't have to use the command line, you can try synaptic command. Previously, in any option we have chosen we need Internet access, so we need to configure it now. First of all, from root directory, execute:

- *mount -t proc proc /proc*

The proc directory is necessary when we want Internet access. It is a pseudo-filesystem used as an interface to kernel data structures. It is commonly mounted at */proc*. Most of it is read-only, but some files allow kernel variables to be changed.

We can remove now all packages we want. Each package and all packages dependant on will be erased with the command

- *apt-get remove – purge package*

The *purge* option is very advisable, in this way apt command ask us for confirmation, showing us all packages will be removed.

We will uninstall all languages packages (i18n), openoffice suite, and all software we don´t consider essential. Remember we are very limited with space on disk.

We advice you changing the Debian package resources lists located in */etc/apt/sources.list* and choose only one packages repository. In my opinion, the better choice is the next one:

deb http://ftp.us.debian.org/debian/ stable main contrib non-free

In order to see enlarged the installed packages then this command will list the packages with size in descending order:

- *dpkg -query -W - -showformat= '${InstalledSize} ${Package} \n ' | sort n*

At least we need install the *gnome-terminal* and *uml-utilities* packages.

- *apt-get install gnome-terminal uml-utilities*

Other interesting command is

- *deborphan*

It will list all packages that have no packages depending on them.

One apt option is *apt-get upgrade*, but we strongly disadvice you using it, because it can stop the CD working.

Before starting the vnuml install, we create the directory /usr/local/bin to avoid later troubles (vnuml needs this directory to be successfully installed).

- *mkdir /usr/local/bin*

and then we start to install it as usual.

A.2.2 Setting user environment

If we want to customize the user environment, we will follow theses commands (it´s necessary the proc partition to be mounted):

- *chmod 777 /tmp*
- *mv /etc/skel /home/knoppix*
- *chown -R knoppix.knoppix /home/knoppix*
- *su - knoppix*
- *startx*

It is important changing the permissions of /tmp to full read/write and execute. If you do not, then KDE will not work.

Sometimes this step fails due to the fact that x server is not well set up. In my case, if I have any problem, I copy the /etc/X11/XF86Config-4 file of my installed Debian to the chroot environment, and startx command won´t fail again.

From the Xserver, we modify all we want, removing icons, changing menu, or adding new icons desktop. When the desktop was like we want, we go out the X session using KDE menu, logout option. It is suggested that you exit KDE closing session, drop back to the command prompt an rerun startx to make sure all changes are applied.

When you are happy with all the changes you have made, you have to exit KDE closing session and return to console, exit knoppix user session, and you will be root again. Then you overwrite /etc/skel file, which is where the environment user is loaded when Knoppix boots:

- *rm -rf /etc/skel*
- *mv /home/knoppix /etc/skel*
- *chown -R root.root /etc/skel*
- *rm -rf /home/knoppix*

You have to change again the /tmp permissions back to read and execute. And you have to remove the x server configuration file too. This file has to be created in each CD boot.

- *chmod 555 /tmp/*

- *rm /etc/X11/XF86Config-4*

It´s the moment to mention the /etc/X11/Xsession.d/45xsession file. This scripts controls how Knoppix behaves and how Knoppix creates the knoppix user´s home directory. So we edit the /etc/X11/Xsession.d/45xsession file, and go to line 64, changing it as follows:

- *rsync -Ha ignore-existing /etc/skel/ $HOME/ 2>/dev/null*

A.3 Kernel remastering

We are going to study now the recompiling kernel process, main aim of this document.

In the boot process, first of all Knoppix will load the kernel placed at /mnt/hda1/knx/master/boot/isolinux/linux using the isolinux boot handler. Later it will look for a boot disk, in this case it will be /mnt/hda1/knx/master/boot/isolinux/minirt.gz. It will uncompress that file, establishing the symbolic links needed to boot, an then it will look for the necessary modules. If we boot from CD drive (not an USB one, for example), the necessary modules are *cloop.ko*, in order to uncompress the CD image, and *unionfs.ko* module, in order to use the filesystem.

A.3.1 Applying kernel patches

We start downloading the version of kernel sources we want from kernel.org, using wget command. In our case, we choose the 2.6.13 version.

- *wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.13.tar.bz2*

From /usr/src directory, and being root user, we copy the default configuration kernel included in Knoppix CD:

- *cd /usr/src*
- *cp linux-2.6.13/.config linux.conf*

We remove the following directory included in Knoppix default CD to avoid problems:

- *rm -rf linux-2.6.12*

And we move here the sources previously downloaded, untaring it :

- *tar -xvjf linux-2.6.13.tar.bz2*

In order to compile the kernel successfully, we need use gcc-2.95 compiler. See the README file located at /usr/src/linux/README and look for the gcc version recommended for compile your kernel version. If we don´t, the compilation process will fail. This gcc version is not included in Knoppix default software, so we are going to install it and setting up like the gcc default version:

- *apt-get update*
- *apt-get install gcc-2.95*
- *cd /usr/bin*
- *rm gcc*
- *ln -s gcc-2.95 gcc*

We create the following symbolic link:

- *cd /usr/src*
- *cd linux/include*
- *ln -s asm-i386 asm*

We download the skas patch,

- *wget http://www.user-mode-linux.org/ blaisorblade/patches/skas3-2.6/skas-2.6.13-v9-pre7/skas-2.6.13-v9-pre7.patch.bz2*

We copy it to /usr/src/linux, from we will execute the following instructions in order to apply it the patch and copy the configuration :

- *bunzip skas-2.6.13-v9-pre7.patch.bz2*
- *patch -p1 < skas-2.6.13-v9-pre7.patch*
- *cp ../linux.conf .config*

We execute this command to complete the config, choosing new modules or doing any change we need

- *make menuconfig*

Or we can exit from this ncurses interface without any change.

Finally we apply the knoppix kernel patch. It is very important for this step to be the last one, because this patch overwrite files that are changed by *make menuconfig* command. And we can start the kernel compilation:

- *patch -p1 <../knoppix-kernel.patch*
- *make && make modules_ install*

We copy the new kernel and System.map files in the correct path:

- *cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.13-skas*
- *cp System.map /boot/System.map-2.6.13-skas*
- *cd /boot*
- *rm System.map*
- *rm vmlinuz*
- *ln -s vmlinuz-2.6.13-skas vmlinuz*
- *ln -s System.map-2.6.13-skas System.map*
- *cd /*
- *rm vmlinuz*
- *ln -s /boot/vmlinuz-2.6.13-skas vmlinuz*

We proceed then to download the source of cloop and unionfs modules, and we will untar them:

- *apt-get install unionfs-source cloop-src*
- *cd /usr/src*
- *tar -xvzf cloop.tar.gz*
- *tar -xvjf unionfs-source.tar.bz2*

With unionfs module we will proceed in the following way:

- *cd modules/unionfs*
- *vi Makefile*

We checks the flags options, and we change the *-g* option by *-DNODEBUG*:

$$UNIONFS\_DEBUG\_CFLAG = \text{-}DNODEBUG$$

This change makes unionfs.ko reduces notably its size. If we don´t do this, we will have lots of problems adding it to minirt.gz.

We compile them with the following command:

- *cd /usr/src/linux*
- *make-kpkg modules_image*

It is possible that union_fs does not compile successfully (not happened to me). In that case, we need to do:

- *cd modules/unionfs*
- *vi Makefile*
- Comment the line where *KVERS* appears, writing: *KVERS=2.6.13-skas3-v9-pre7*
- *make*
- *make install*

We will install the cloop module too (and the unionfs will be installed in the same way if it was compiled successfully doing *make-kpkg modules_image* before):

- *cd /usr/src*
- *dpkg -i cloop**

Then we will umount the proc directory, remove the .deb files in order to save space, and we leave the chroot environment:

- *umount /proc*
- *apt-get clean*
- *we leave chroot (ctrl+D)*

We have to remember to remove our history command:

- *rm -rf /mnt/hda1/knx/source/KNOPPIX/root/.bash_history*
- *rm -rf /mnt/hda1/knx/source/KNOPPIX/etc/skel/.bash_history*

If we don't remove the kernel sources, the CD image will be generated will not fill in a CD, so we recommend remove them when we are sure the kernel works succesfully, and any new software we want to install will not depend on it (For example, some Perl modules necessary for VNUML depend on some files included in kernel sources).

We overwrite the new kernel will be load by CD:

- *cp /mnt/hda1/knx/source/KNOPPIX/boot/vmlinuz-2.6.13-skas /mnt/hda1/knx/master/boot/isolinux/linux*

If we want to modify the initial message editing /mnt/hda1/knx/master/boot/isolinux/boot.msg file.

A.3.2 Modifying minirt.gz

One process more is necessary in order to get the CD boots succesfully, and this one is the need of modifying the initial disk ram compressed, minirt.gz file. We will make a new directory doing these necessary steps:

- *mkdir /mnt/hda1/knx/mroot*

We execute the commands:

- *cd /mnt/hda1/knx*
- *cp master/boot/isolinux/minirt.gz*
- *gunzip minirt.gz*
- *mount -o loop -rw minirt mroot/*
- *cd mroot*

In this directory it is necessary to include the boot scripts, the index.html file is loaded when CD boots, the new kernel, and the new modules. It´s very important to copy them to the CD boot successfully. These modules will be copied from /mnt/hda1/knx/source/KNOPPIX/lib/modules/2.6.13-skas3-v9-pre7 directory, replacing all modules included in modules/scsi directory. The most important ones are cloop.ko and unionfs.ko, the CD will not boot without them.

- *cp /mnt/hda1/knx/source/KNOPPIX/lib/modules/2.6.13-skas3-v9-pre7/kernel/fs/unionfs.ko /mnt/hda1/knx/mroot/modules/*

- *cp /mnt/hda1/knx/source/KNOPPIX/lib/modules/2.6.13-skas3-v9-pre7/ kernel/drivers/block/cloop.ko /mnt/hda1/knx/mroot/modules/*

We remove the cloop.o module included, saving more space.

We can modify too the linuxrc script, in order to changing details as modify the wellcome phrase by our own message, as *Welcome to the Knoppix live Linux-on-CD with VNUML.*

When we have finished all necessary changes, we will overwrite the new minirt.gz

- *cd /mnt/hda1/knx*
- *umount mroot*
- *gzip -9 minirt*
- *cp minirt.gz master/boot/isolinux/minirt.gz*

It is possible to make changes in master/boot/isolinux/isolinux.cfg file, to the default speech of the CD was spanish, for example, replacing *lang=us* by *lang=es*

We will change the index.html file showed when the CD boot by the VNUML html files hosted at http://jungla.dit.upm.es/vnuml at */mnt/hda1/master/index.html.*

A.4 Final CD-ROM Image

We may now make the CD-Rom Image, with the following instructions:

- *rm -rf /mnt/hda1/knx/source/KNOPPIX/.rr_ moved*
- *mkisofs -R -U -V "KNOPPIX.net filesystem" -publisher "KNOPPIX www.knoppix.net" -hide-rr-moved -cache-inodes -no-bak -pad /mnt/hda1/knx/source/KNOPPIX | nice -5 /usr/bin/create_ compressed_fs - 65536 > mnt/hda1/knx/master/KNOPPIX/KNOPPIX*
- *cd /mnt/hda1/knx/master; find -type f -not -name md5sums -not -name boot.cat -not -name isolinux.bin -exec md5sum '{}' \; > KNOPPIX/md5sums*
- *mkisofs -pad -l -r -J -v -V "KNOPPIX" -no-emul-boot -boot-load-size 4 -boot-info-table -b boot/isolinux/isolinux.bin -c boot/isolinux/boot.cat -hide-rr-moved -o /mnt/hda1/knx/knoppix.iso /mnt/hda1/knx/master*

Before you burn the ISO, you can test it with virtualization tools like qemu or VMware, verifying all works like you want.

Bibliografía

[1] http://www.knoppix.net

[2] Shon Harris, *All in One CISSP*, Mc Graw Hill, 2005.

[3] http://user-mode-linux.sf.net

[4] Angela D. Orebaugh, *Ethereal Packet Sniffing*, Sybgress Publishing, 2004

[5] Marc J. Rochkind, *Advanced UNIX Programming*, Second Edition, 2005

[6] http://www.tcpdump.org/pcap/pcap.html

[7] http://gtk2-perl.sourceforge.net/doc/intro/

[8] http://linux.omnipotent.net/article.php?article_id=12504

[9] W. Richard Stevens, *Unix Network Programming*, Prentice Hall, 1998

[10] M. Tim Jones, *GNU Linux Application Programming*, Charles River Media, 2005

[11] Fermín Galán Marquez, *Tecnología XML en la herramienta de simulación de redes VNUML*, 2004

[12] http://www.winpcap.org/docs/iscc01-wpcap.pdf

[13] http://www.tcpdump.org/papers/bpf-usenix93.pdf

[14] http://www.hax-linux.org

[15] http://slax.linux-live.org/.

[16] http://geexbox.org.

[17] http://www.am-utils.org/project-unionfs.html

[18] http://www.kernelthread.com/publications/virtualization/

[19] http://www.devx.com/Intel/Article/30125

[20] http://www.linuxjournal.com/article/8540

[21] http://www.itarchitect.com/shared/article/showArticle.jhtml?articleId=172302134&classroom=

[22] http://www.vmware.com

[23] http://www.linuxelectrons.com/article.php/20050214152005354

[24] http://www.infineon.com/tpm

[25] http://openvz.org

[26] http://www.microsoft.com/windows/virtualpc/default.mspx

[27] http://www.microsoft.com/windowsserversystem/virtualserver/default.mspx

[28] http://usenix.org/events/osdi02/tech/full_papers/whitaker/whitaker_html/node7.html

[29] http://home.gna.org/adeos/

[30] http://fabrice.bellard.free.fr/qemu/

[31] http://user-mode-linux.sourceforge.net

[32] http://squashfs.sourceforge.net/

[33] http://freshmeat.net/projects/zisofs-tools/

[34] http://en.wikipedia.org/wiki/Cloop

[35] http://linuxvalley.it/encyclopedia/meteokernel/kernel24/doc24/filesystems/cramfs.txt

[36] http://lilo.go.dyndns.org/

[37] http://www.gnu.org/software/grub/

[38] http://syslinux.zytor.com/

[39] http://syslinux.zytor.com/iso.php

[40] http://www.gtk.org

[41] http://www.gnome.org

[42] http://www.gimp.org

[43] http://gtk2-perl.sourceforge.net