- PSYNC -

Sincronización de un entorno virtual para ser usado por usuarios con necesidades de movilidad

Alumno: Luis Somolinos Valdiviejas
Grado de ingeniería Informática

Consultor: Francesc Guim Bernat

23 de enero de 2012

Índice de contenido

Índice de ilustraciones

Glosario

CPD: Centro de Procesamiento de Datos. Suele ser un edificio o sala de gran tamaño usada para mantener en él una gran cantidad de equipamiento electrónico. habitualmente son creados y mantenidos por las organizaciones con objeto de tener acceso a la información necesaria para sus operaciones.

Driver: Programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz, posiblemente estandarizada, para usarlo.

Freeware: Define un tipo de software que se distribuye sin costo, disponible para su uso y por tiempo ilimitado.

Hypervisor: Tecnología compuesta por una capa de software que permite utilizar, al mismo tiempo, diferentes sistemas operativos o maquinas virtuales en un mismo ordenador físico.

Mainframe: Computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos.

Nube (*Cloud*): Es un paradigma mediante el cual los sistemas informáticos se ofrecen como servicios en lugar de como producto. De esta forma, el almacenamiento, la información, etc. son ofrecidos a los puestos finales (y otros dispositivos) como meros servicios que se dan a través de la red (generalmente internet).

Servidores x86: Servidores basados en microprocesadores x86 que es la denominación genérica dada a ciertos microprocesadores de la familia Intel, sus compatibles y la arquitectura interna básica que estos procesadores poseen. Han constituido desde su nacimiento un estándar para los ordenadores del tipo Compatible IBM PC.

Shebang: Es el nombre que recibe el par de caracteres "`#!`" que se encuentran al inicio de los programas ejecutables interpretados. A continuación de estos caracteres se indica la ruta completa al intérprete de las órdenes contenidas en el mismo (un ejemplo habitual de *shebang* sería "`#!/bin/bash`", que es una llamada al intérprete de comandos *Bash*). Este método permite que el usuario pueda ejecutar un programa interpretado de la misma forma que ejecutaría un programa binario.

TIC: Tecnologías de la Información y la Comunicación. Estas tecnologías agrupan los elementos y las técnicas usadas en el tratamiento y la transmisión de las informaciones, principalmente de informática, internet y telecomunicaciones. Por extensión, designan el sector de actividad económica.

1. MOTIVACIÓN

Hoy en día estamos asistiendo a un incesante bombardeo del concepto de virtualización, de tal forma que, las grandes empresas del sector pueden virtualizarnos nuestros puestos de trabajo, las aplicaciones, el almacenamiento, los servidores, el CPD y hasta nuestro departamento de IT entero si decidimos engancharnos a eso que se ha dado en llamar la "Nube" (*Cloud*). Sin embargo, hace apenas 10 años, sólo unos pocos tenían claro que era eso de virtualizar.

La virtualización surgió como una idea de visionarios universitarios en los años 60. Posteriormente, durante los años 70 y 80, IBM aplicó estas teorías principalmente al mundo de los mainframe creando máquinas físicas cuyos recursos hardware podían repartirse de forma que se asignaban a máquinas virtuales consiguiéndose, de esta forma, un aprovechamiento mayor de los recursos hardware totales. Esta tendencia, seguida también por empresas como SUN con sus servidores SPARC, implicaba la compra de grandes máquinas físicas, muchas veces con tecnologías propietarias y de elevados costes, por lo que, en realidad, la virtualización no empezó a hacerse popular hasta el 2001, cuando la empresa VMware sacó al mercado su hypervisor para sistemas x86 (muy extendidos en la época gracias a su sencillez de gestión, mantenimiento y a que tenían un precio muy competitivo con respecto a los mainframes).

Como siempre, los comienzos no son fáciles y las grandes empresas se resistían a cambiar su forma "tradicional" de trabajar. Así, en 2008 (¡Hace sólo 3 años!), todavía se estimaba que apenas el 12% de la carga de trabajo de servidores x86 estaba ejecutándose sobre máquinas virtuales (Gartner 2010 [2]).

Actualmente el impacto de la virtualización en el sector de las TIC es innegable. Se podría afirmar, sin temor a equivocarse, que no hay un trabajador del sector que no haya oído hablar de la virtualización. No en balde, se estima que la carga de trabajo de servidores x86 que las empresas dejarán en manos de máquinas virtuales en el año 2012, rondará el 50% (Gartner 2010 [2]).

El beneficio de esta tecnología para las empresas es inmediato ya que se reducen los costes asociados al hardware (no pensemos sólo en la compra sino también en gastos asociados a mantenimiento, electricidad, refrigeración, etc.) gracias a la "consolidación de servidores". Esta técnica consiste en utilizar la virtualización para repartir los recursos de un servidor físico entre varias máquinas virtuales de tal forma se pueden tener varios servidores virtuales ejecutándose en un mismo servidor físico. La consecuencia inmediata es la reducción del número de servidores físicos necesarios en el CPD, con el consiguiente ahorro económico que se mencionaba anteriormente.

Aunque, como se ha explicado, la consolidación de servidores presenta un beneficio inmediato para la empresas y, de hecho, éste suele ser el principal aliciente para las empresas a la hora de adoptar la virtualización (VMware 2011 [1]), los beneficios no se limitan a la consolidación y, yendo más allá, se pueden conseguir cosas como: un aumento en la facilidad de la gestión del CPD, consolidación de los recursos de

almacenamiento, mejora y simplificación del los mecanismos de alta disponibilidad de los sistemas, mejora del mantenimiento del puesto cliente (con la virtualización del puesto cliente o VDI), mejora en el aprovisionamiento de nuevos servidores para proyectos (Cloud privada), externalización completa del CPD (Cloud pública), etc.

Está fuera del alcance del presente documento entrar a explicar los cómo y porqué de cada uno de los citados beneficios. Lo que sí parece haber quedado claro es que la virtualización ha entrado en el sector de las TIC por méritos propios y que es una opción que las empresas deberán tener muy presente de ahora en adelante.

Pero... ¿Qué ocurre con los usuarios particulares?. En párrafos anteriores se ha hablado de consolidación de servidores, facilidad para la gestión de grandes CPD, virtualización del puesto cliente, etc. Esta claro que todos esos beneficios no se pueden aplicar a usuarios particulares cuyos “activos” no suelen pasar de dos o tres PC (en el mejor de los casos). Siendo así, ¿Es la virtualización una tecnología sólo útil para empresas o, por el contrario, pueden obtener beneficio de ella los usuarios particulares?

Como no podía ser de otra forma, la respuesta a la pregunta anterior es que los usuarios particulares también pueden beneficiarse de una tecnología tan versátil como la virtualización. Evidentemente un usuario particular no podrá aspirar a obtener grandes beneficios económicos pero si algunos beneficios funcionales como, por ejemplo, la posibilidad de poder usar máquinas virtuales con sistemas operativos diferentes sin tener por ello que disponer de varios PC físicos. Algo que comienza a ser habitual, sobre todo, porque suele ser sencillo encontrar hypervisores (incluso de la empresas punteras del sector) con licencia gratuita.

Otros ejemplos de utilidad que la virtualización podría ofrecer a un usuario final son:

- Probar configuraciones: Algunas veces, intentando aplicar una nueva configuración al sistema o, simplemente, instalando alguna aplicación de “dudoso origen”, lo que conseguimos es dejar inestable nuestro PC. En el peor de los casos se deberá recurrir a una reinstalación del sistema, con la consiguiente pérdida de tiempo y quizás de datos, ya que, en el entorno particular, las políticas de backup no siempre son tan adecuadas como debiera. Las máquinas virtuales pueden ayudar a solucionar este problema, gracias a lo rápido y fácil que es montar una de estas máquinas virtuales.

  Por ejemplo, un usuario cualquiera, podría crear una máquina virtual donde realizar la pruebas (de configuración o instalación), una vez verificado que el cambio merece la pena se puede aplicar sobre el sistema principal. En caso contrario, bastará con borrar la máquina virtual y el sistema principal seguirá tan estable como siempre.

- Sand-box para internet: Todos sabemos lo difícil que es mantener “limpio” un equipo conectado a internet. Ni siquiera con el mejor de los antivirus estamos cien por cien seguros. La facilidad con la que se clonan las máquinas virtuales (generalmente sólo implica copiar unos pocos ficheros) también puede ayudarnos en este caso.

Por ejemplo, bastaría con crear una máquina virtual con nuestras herramientas favoritas para navegar. Una vez todo configurado, clonaríamos esta máquina y la utilizaríamos para navegar. Al finalizar, la podríamos destruir sin problemas.

Cuando necesitásemos navegar nuevamente, obtendríamos un nuevo clon y, al arrancarlo, podríamos navegar con la certeza de partir de nuevo con una máquina impecable.

- Otros... Los anteriores son sólo algunos ejemplos, sin embargo, la virtualización es una tecnología muy versátil y puede ser aplicada en infinidad de escenarios. Depende de la imaginación del usuario.

En este sentido, la motivación del presente proyecto será poner de manifiesto la utilidad de la virtualización, ya no para grandes empresas que cuentan con grandes CPD, sino para usuarios particulares que también pueden obtener beneficios inmediatos de la utilización de máquinas virtuales. Para ello, se planteará una situación cotidiana en la vida de una usuario particular y se intentará resolver gracias a las funcionalidades que nos ofrece la virtualización.

En el caso concreto que se desarrollará a lo largo del presente trabajo, se utilizará la virtualización para solucionar el caso particular de usuarios que, realizando su trabajo normalmente en un puesto, necesitan poder seguir trabajando a pesar de tener que viajar y no poder acceder a su puesto de trabajo habitual.

Como solución se propondrá la creación de una máquina virtual con las mismas características que el puesto de trabajo habitual y se creará un programa que permita mantener sincronizados los entornos de trabajo, tanto del puesto habitual, como de la máquina virtual. De esta forma, el usuario, en el momento de salir de viaje, podrá sincronizar su puesto habitual con la máquina virtual, salir de viaje y trabajar con ésta (posiblemente montada en un portátil) y, a la vuelta, sincronizar de nuevo la máquina virtual con el puesto habitual para volcar todo el trabajo realizado durante el viaje.

2. OBJETIVOS

Como ya se ha mencionado en el apartado anterior, el objetivo del presente trabajo será aplicar la tecnología de la virtualización en un supuesto cotidiano para un usuario informático particular.

Concretamente, como hipótesis de trabajo, asumiremos el caso de un usuario que utiliza su PC de sobremesa particular como herramienta para su trabajo. Este usuario, que también se dedica al mundo de la informática, utilizará el citado PC de sobremesa para realizar desarrollos, elaborar documentación, etc.

Por otro lado, nuestro usuario se ve obligado, por motivos laborales, a realizar frecuentes viajes que le mantienen alejado de su domicilio habitual durante varios días e incluso semanas. Debido a estos viajes, el usuario de nuestro ejemplo siempre se retrasaba en la entrega de sus proyectos ya que no podía trabajar mientras estaba fuera de su domicilio.

Para solucionar este problema, el usuario decide comprarse un ordenador portátil con la esperanza de poder adelantar algo de trabajo en los ratos perdidos que tiene entre una y otra reunión de trabajo en sus viajes. Lo primero que observa nuestro usuario en su nuevo portátil es que, la versión de sistema operativo que viene preinstalada, no es exactamente la que el suele utilizar. Sin embargo, no se desanima y se pone a buscar en Internet. En unas pocas horas ha conseguido descargarse e instalar en el portátil las versiones adecuadas de sus compiladores y herramientas ofimáticas favoritas.

En la víspera de su siguiente viaje, nuestro usuario, decide, para no dejar parado su actual proyecto, copiarse en el portátil los tres o cuatro últimos ficheros fuente que ha estado retocando a lo largo de la semana y el documento donde está reflejando los cambios. Esta vez, ¡Nada podrá retrasarle!

Sin embargo, cuando nuestro personaje regresa de su viaje, llega cabizbajo. En realidad no ha podido avanzar nada en su trabajo. Cuando intentó compilar alguno de los ficheros fuente en su portátil, tras varias horas de modificar código, resulta que el proceso de compilación falló porque olvidó copiar unas librerías a las que se hacía referencia desde el código. Además, no pudo modificar el documento porque, su procesador favorito, no funcionaba en el sistema operativo del portátil y, para poder instalarlo, tuvo que poner una versión superior que no reconoció adecuadamente el formato del documento original, desbaratándolo por completo.

Esta claro, que la simple solución de copiar unos cuantos ficheros al tun-tun en el portátil no ha sido una solución adecuada para el usuario de nuestro ejemplo. Por tanto, se asumirá como objetivo del presente proyecto, ofrecer una solución adecuada para la hipótesis desarrollada en párrafos anteriores. Dicha solución se basará en tecnologías de virtualización y deberá cumplir con los requisitos de satisfacer las necesidades del usuario, sin perder de vista, que ha de ser una solución sencilla y asequible ya que deberá estar al alcance de usuarios particulares.

Para cubrir este objetivo, se desarrollará una pequeña aplicación cuya misión será mantener un sincronismo entre el PC de sobremesa (en adelante “Puesto fijo”) y el portátil (en adelante “Puesto móvil”).

Para que la aplicación pueda abstraerse de las peculiaridades del hardware y el sistema operativo del Puesto móvil, se utilizará una máquina virtual (en adelante “Puesto virtual”) que replicará fielmente el sistema operativo del Puesto fijo. De esta forma, lo que realmente quedará sincronizado no será el Puesto móvil en sí mismo, si no, el Puesto virtual que se ejecuta sobre él y por ende, el usuario, realmente, desempeñara su trabajo sobre el Puesto virtual y no sobre el Puesto móvil.

Finalmente, la aplicación, proporcionará el procedimiento adecuado para que, una vez finalizado un viaje, los datos que hayan sido modificados en el Puesto virtual puedan volver a ser volcados de forma sincronizada sobre el Puesto fijo. Permitiéndose así que el usuario pueda continuar trabajando sobre dicho Puesto fijo como si nada hubiese pasado. En ese momento el Puesto virtual podrá ser destruido, si se desea, sin mayores consecuencias.

Merece la pena mencionar que, a pesar de que los objetivos del presente trabajo se centran en un hipotético usuario final, la solución mostrada podría ser fácilmente extrapolada a entornos profesionales con necesidades de desplazamiento. Pensemos, por ejemplo, en empresas con grupos de trabajo de ingenieros que deban desplazarse “sobre el terreno”. Para estas empresas es un verdadero despilfarro de recursos el hecho de que sus ingenieros no puedan trabajar hasta que vuelven de sus viajes. Sin embargo, la solución que se propone, supondría un gran aumento de productividad ya que, los ingenieros, podrían aprovechar para adelantar trabajo también durante los viajes.

3. DESCRIPCIÓN

Con el fin de cubrir objetivo marcado, se instalará un entorno de laboratorio que simulará la situación explicada en la hipótesis de trabajo del apartado anterior. Los elementos hardware que constituirán este laboratorio serán:

- Un ordenador PC de sobremesa que representará el papel de Puesto fijo. Dicho PC posee las siguientes características:
  - Procesador: Intel Pentium IV a 3.2 GHz
  - Memoria: 2 GiB RAM
  - Disco duro: 160 GiB
  - Sistema operativo: Ubuntu 10.04 (32 bits)
- Un ordenador PC tipo portátil, representando el papel de Puesto móvil, con las siguientes características:
  - Procesador: Intel Centrino Core 2 Duo a 2.1 Ghz
  - Memoria: 4 GiB RAM
  - Disco duro: 320 GiB
  - Sistema operativo: Windows Vista Home Premium SP2 (32 bits)

Además de estos equipos físicos, será preciso contar con una capa de virtualización. Ésta se conseguirá instalando un hypervisor VMware Server 2.0 en el Puesto móvil. Sobre dicho hypervisor se ejecutará el Puesto virtual que tendrá las siguientes características:

- Procesador: 1 vCPU
- Memoria: 2 GiB vRAM
- Disco duro: 10 GiB vDisk
- Sistema operativo: Ubuntu 10.04 (32 bits)

Para terminar con el entorno del laboratorio, se utilizará un intérprete de Perl (v5.10.1), instalado en las máquinas Linux, para permitir la ejecución de los *scripts* que formarán la aplicación que se desarrollará a lo largo del presente trabajo y que deberá ser capaz de realizar las siguientes tareas:

a) Determinar qué directorios y ficheros, de los utilizados en el trabajo realizado habitualmente en el Puesto fijo, son realmente necesarios replicar al Puesto virtual.

b) Generar, a partir de la información obtenida en el paso a), un fichero único, en formato tgz, que contenga toda la información necesaria para realizar un volcado de los datos necesarios en el Puesto virtual. La información necesaria será aquella que permita al usuario continuar con

su trabajo en el Puesto virtual.

En este punto se deberá crear el Puesto virtual que se podrá obtener, bien, creando una nueva máquina virtual desde cero o desplegando una plantilla.

c) Una vez creado el Puesto virtual, se copiará, en dicho Puesto, el fichero creado en el paso b) y se lanzará un proceso que volcará los datos adecuados sobre el Puesto virtual.

En este punto, el usuario, podrá desempeñar su trabajo sobre el Puesto virtual sin problema.

d) Una vez terminado el trabajo del usuario sobre el Puesto virtual, se ejecutará un nuevo proceso que será capaz de extraer los datos adecuados del Puesto virtual para almacenarlos, una vez más, en un fichero único en formato tgz.

e) Por último, el fichero obtenido en el paso anterior, deberá ser copiado al Puesto fijo donde se ejecutará un nuevo proceso que volcará, los datos modificados en el Puesto virtual, sobre los directorios originales del Puesto fijo.

En este último punto del proceso, el usuario habrá podido trabajar fuera de su Puesto fijo, utilizando el Puesto móvil. Además, todos los cambios que hubiese realizado sobre los ficheros del Puesto virtual, se habrán volcado sobre el Puesto fijo, por lo que podrá continuar su trabajo, en el punto en el que lo dejó, en su Puesto virtual.

4. PLANIFICACIÓN

La planificación para la ejecución del presente proyecto, estimada en un principio, contempla la realización del trabajo diferenciando dos grandes tareas:

- Desarrollo: engloba todas aquellas subtareas relativas al trabajo de creación e implementación de la solución software que se pretende obtener.

- Memoria: Esta tarea contiene la planificación de subtareas relativas a la obtención de la documentación que será necesario entregar al final del proyecto (principalmente, la memoria de proyecto).

El diagrama de Gantt que representa la planificación inicial, se muestra en la ilustración 1.

Como se puede observar en dicho diagrama, las tareas de “Desarrollo” y “Memoria” se llevan a cabo en paralelo. Esto es así porque, la memoria, necesariamente, irá evolucionando a partir de los datos que se obtengan del diseño y la implementación de la solución software del proyecto.

En lo que respecta a la tarea de Desarrollo, se ha dividido en las siguientes subtareas:

Tarea	Descripción	Fecha inicio	Fecha fin	Previsión horas
Búsqueda de documentación	Búsqueda y recopilación de información (tutoriales, comparativas, artículos, etc.) relativas a las tecnologías que se utilizarán en el proyecto.	26/09/11	04/11/11	45
Montaje del laboratorio	Trabajos de montaje de la infraestructura del laboratorio donde se creará y probará la solución software que se desea obtener.	24/10/11	07/11/11	17
Diseño	Fase de diseño de la solución.	31/10/11	21/11/11	24
Implementación	Desarrollo de todo el código necesario.	22/11/11	01/01/12	45
Pruebas	Pruebas que permitirán validar que la solución software cumple con los requisitos esperados.	26/12/11	16/01/12	24

Por otro lado, la tarea de elaboración de la memoria no requiere de mayor explicación ya que podría haberse planteado como una única tarea continua. La separación en subtareas corresponde, únicamente, al hecho de representar las distintas entregas parciales que deberán hacerse del documento final.

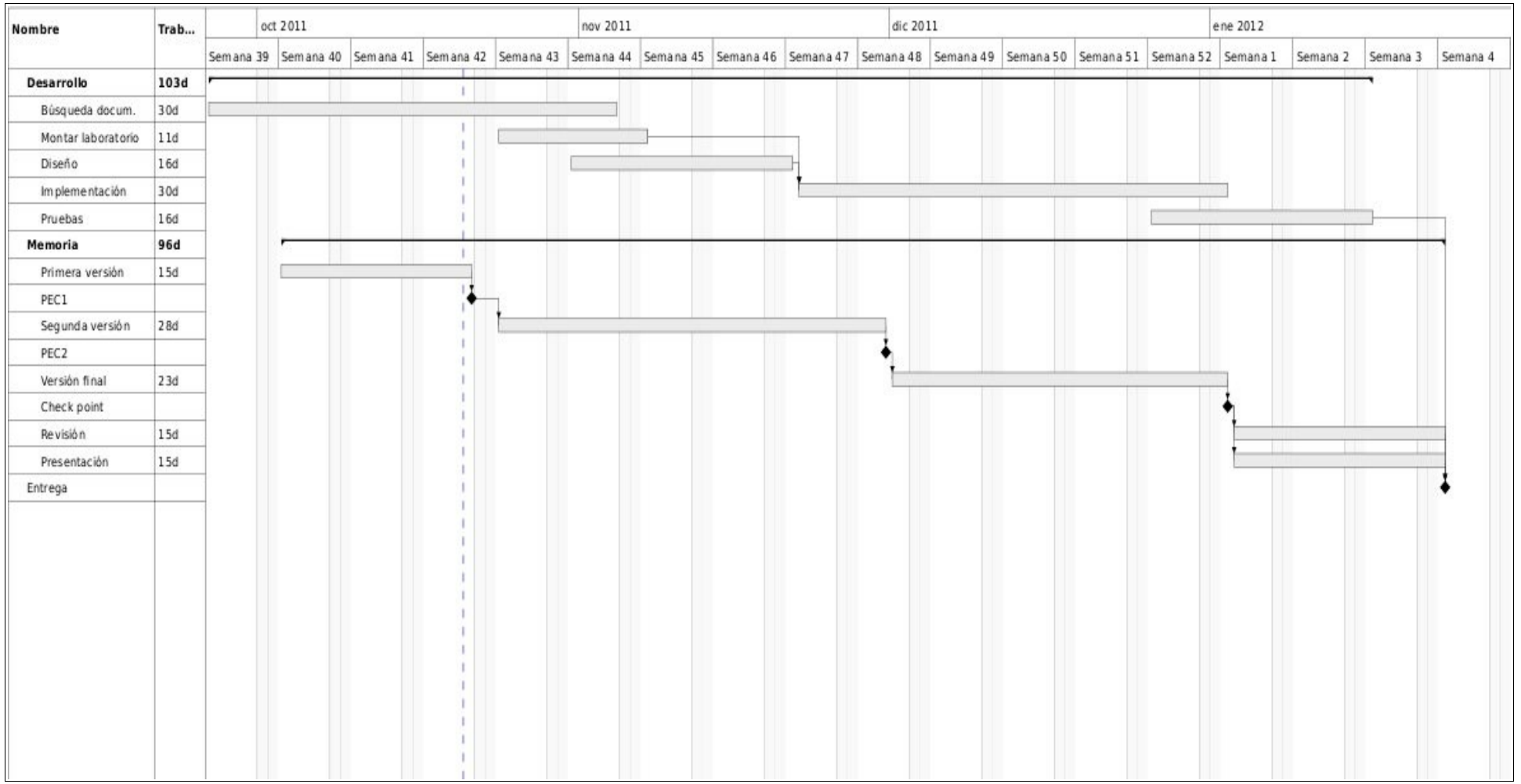

Ilustración 1: Diagrama de Gantt del proyecto.

5. VIRTUALIZACIÓN

Como ya se ha mencionado en el apartado 3 (DESCRIPCIÓN). En el presente proyecto se utilizará un hypervisor que nos aportará la capa de virtualización que permitirá la ejecución del Puesto Virtual. Se ha elegido la utilización del hypervisor VMware Server 2.0.

En el presente apartado se pretende dar un repaso a las características de las principales soluciones de virtualización del mercado. El objetivo no es la elaboración de una precisa lista de productos comerciales y la descripción detallada de cada una de sus funcionalidades si no, más bien al contrario, se tratará de explicar las características, arquitectura y principales ventajas e inconvenientes de las soluciones que se pueden encontrar en el mercado para que el lector pueda hacerse una idea del “estado del arte” de esta tecnología.

Finalmente, una vez presentados los distintos tipos de hypervisores, se justificará la decisión tomada de utilizar VMware Server.

5.1. Tipos de Hypervisores

Como se ha mencionado con anterioridad, en estos momentos parece que se está produciendo un boom en el ámbito de la virtualización, de tal forma que cuesta estar al día de todas las tecnologías presentes en el mercado. A pesar de las funcionalidades propias de los productos de cada fabricante, que hacen que cada solución sea diferente de las otras, podemos encontrar una serie de características comunes que permiten clasificar a los hypervisores en lo siguientes grupos [6].

5.1.1. Hypervisores tipo *Hosted*

Estos hypervisores se instalan sobre un sistema operativo, es decir, el equipo físico sobre el que se instalan tiene que tener previamente instalado un sistema operativo de propósito general (Linux, Windows, Mac, etc.).

Por tanto, este tipo de hypervisores se instalan sobre el sistema operativo como una aplicación más y, cuando se ejecutan, se crea la capa de virtualización sobre la que corren los sistemas operativos de las máquinas virtuales, a los que se suele llamar sistemas operativos *guest* (huésped).

El siguiente esquema lógico muestra la arquitectura descrita.

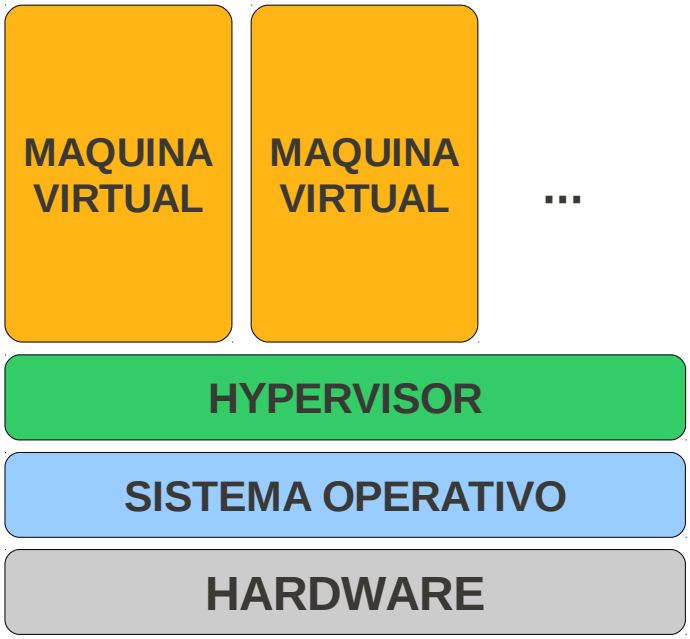

Ilustración 2: Esquema general de un hypervisor tipo Hosted.

La principal ventaja de este tipo de hypervisores es que ofrecen una gran compatibilidad con el hardware. Esto se debe a que, el hypervisor, no interactúa directamente con el hardware de la máquina física, si no con la abstracción de dichos dispositivos que presenta el sistema operativo *host*. De esta forma, si el dispositivo hardware es correctamente detectado y se cuenta con unos drivers adecuados que permitan su correcto funcionamiento en el sistema operativo *host*, con casi toda certeza, el dispositivo podrá ser utilizado en el sistema *guest*.

Por otro lado, su principal inconveniente es precisamente el sistema operativo *host* ya que, al ejecutarse como una aplicación más, deben compartir recursos con él, lo que provoca rendimientos bajos en las máquinas virtualizadas. Además, a nivel de seguridad, el sistema operativo *host* es un punto único de fallo, ya que, un ataque sobre dicho sistema (o cualquier otro incidente) que provoque el bloqueo del sistema operativo *host*, provocará, a su vez, el bloqueo de todas los procesos que se ejecuten sobre él, incluido el hypervisor. Esto supondrá, también, el bloqueo de todas las máquinas virtuales que se estuviesen ejecutando sobre dicho hypervisor.

Ejemplos de este tipo de Hypervisores son:

- VMware Server [9]
- VMware Workstation [10]
- Microsoft Virtual Server [11]

5.1.2. Hypervisores tipo *Bare metal*

Estos hypervisores se instalan directamente sobre un equipo, sin necesidad de que exista ningún otro software cargado en él. El siguiente esquema muestra la estructura lógica de estos hypervisores.

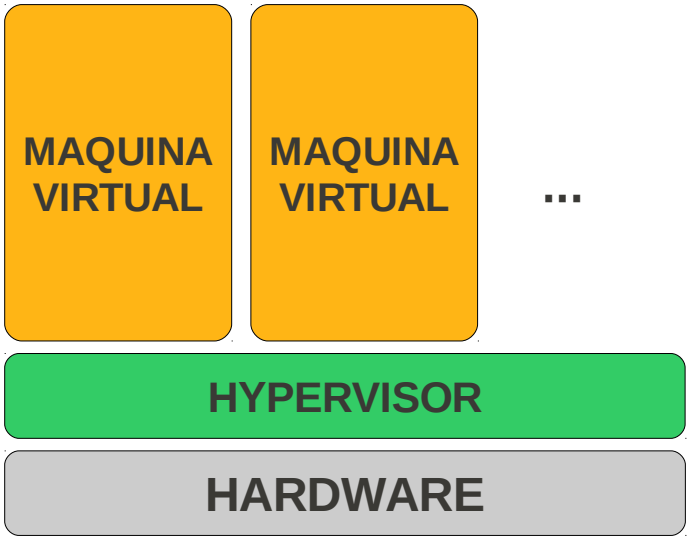

Ilustración 3: Esquema general de un hypervisor tipo Bare metal.

Este tipo de hypervisores, al interactuar directamente con el hardware, evitan los problemas de rendimiento y robustez que tenían los hypervisores vistos anteriormente. Precisamente, por esta razón, este tipo de hypervisores son los recomendados por sus respectivos fabricantes para soportar los entornos de producción.

En cuanto a la cuestión de la compatibilidad con el hardware, depende mucho de cómo se realicen los accesos al hardware físico. Existen dos formas con las que los hypervisores tipo *Bare metal* gestionan el acceso al hardware. Dependiendo de la estrategia de acceso al hardaware, los hypervisores de tipo *Bare metal* se pueden subdividir en.

5.1.2.1. Hypervisores Bare metal monolíticos

En este caso, el hypervisor interactúa directamente con el hardware. Para ello, debe integrar una serie de *divers* nativos diseñados expresamente para que el hypervisor acceda a un tipo concreto de hardware. La siguiente ilustración muestra la estructura descrita.

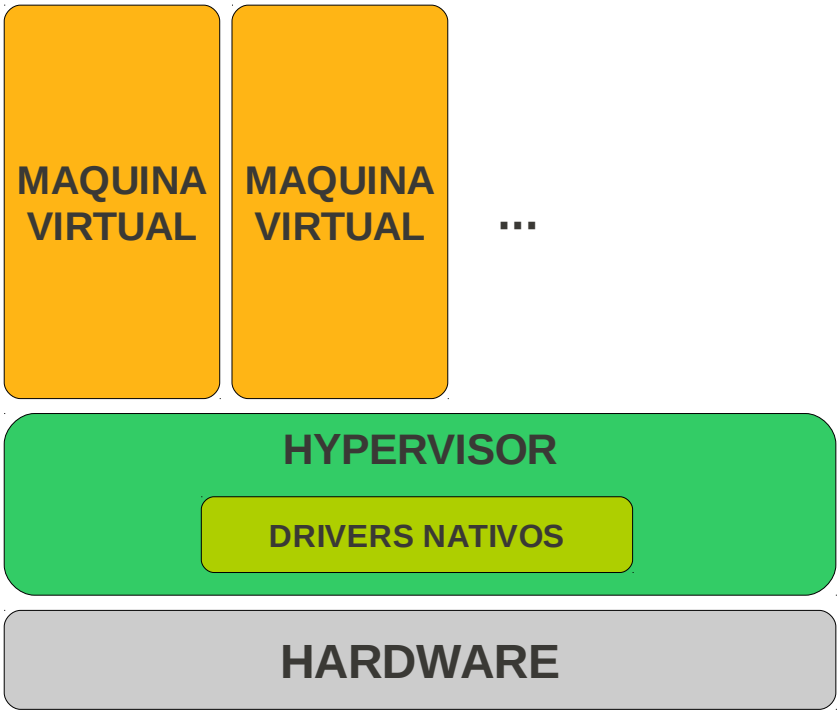

Ilustración 4: Esquema general de un hypervisor Bare metal monolítico.

Este tipo de hypervisores suelen obtener los mejores rendimientos ya que acceden directamente al hardware sin tener la intermediación de particiones o sistemas externos que gestionen el acceso al hardware. Es decir, todos los sistemas operativos *guest* acceden directamente al hardware del dispositivo físico a través de los correspondientes drivers nativos del hypervisor.

Otra ventaja que presentan este tipo de hypervisores es su bajo perfil de instalación, es decir, la poca cantidad de espacio en disco que ocupa uno de estos hypervisores una vez instalado en un servidor físico [15]. La instalación de un hypervisor de tipo monolítico suele rondar unos pocos cientos de MiB, mientras que un hypervisor de tipo microkernel (que veremos más adelante) suele ocupar unos cuantos GiB. Esto es debido a que sólo se deben instalar aquellos componentes que son imprescindibles para el funcionamiento del hypervisor.

Como inconveniente, los hypervisores monolíticos pueden presentar mayores problemas de compatibilidad hardware ya que, sólo pueden funcionar, si incorporan los drivers adecuados para trabajar con el hardware que deseamos utilizar.

El principal ejemplo comercial de este tipo de hypervisores, y uno de los más utilizados actualmente, es el VMware vSphere ESXi [12].

5.1.2.2. Hypervisores Bare metal microkernel

Este tipo de hypervisores implementan, lo que se denomina, una partición padre (*parent*) o raíz (*root*) que es necesario instalar junto con el hypervisor y que actúa como una máquina virtual, cuyo sistema operativo, es quien se encarga de la gestión de los drivers que permiten el acceso al hardware físico.

Estos hypervisores, al contrario que los vistos en el apartado anterior, no necesitan drivers nativos ya que, los drivers necesarios para acceder al hardware, se instalan únicamente en el sistema operativo de la partición padre. Por tanto, los sistemas operativos *guest* nunca acceden directamente al hardware físico. Las peticiones de acceso se traducen, en realidad, en peticiones a la partición padre que es la que gestiona todo el acceso al hardware disponible.

La siguiente ilustración muestra un esquema de la estructura de este tipo de hypervisores:

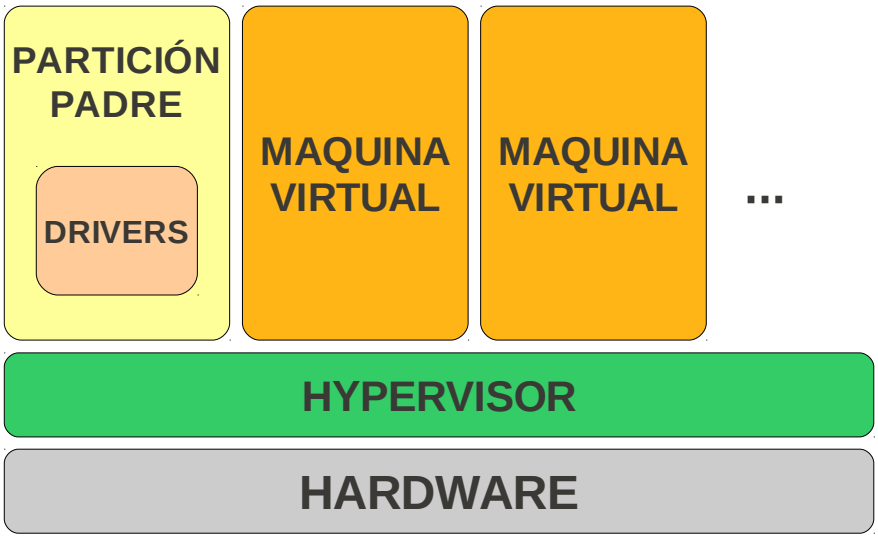

Ilustración 5: Esquema general de un hypervisor Bare metal microkernel.

Este tipo de soluciones suelen mejorar bastante el problema de las incompatibilidades de hardware ya que, en las particiones padre, se instalan sistema operativos de propósito general (normalmente Windows o Linux), lo que permite, de golpe, que el hypervisor pueda trabajar con todos los drivers ya creados para dichos sistemas operativos genéricos.

Como inconveniente, sigue apareciendo el tema del rendimiento, ya que, al tener que ser gestionados todos los accesos al hardware a través de la partición padre, es posible la aparición de “cuellos de botella” en dicha partición. Lo que repercute en el rendimiento de los sistemas *guest*.

Por otro lado, como ya se apuntó en la descripción de los hypervisores monolíticos, las instalaciones de hypervisores tipo microkernel suelen ser bastante más pesadas (del

orden de GiB) ya que, como ahora se puede ver, en la instalación de un hypervisor microkernel se necesita instalar, no solamente el hypervisor propiamente dicho, si no también un sistema operativo completo en la partición padre.

Ejemplos comerciales de este tipo de hypervisores son:

- Citrix XenServer [13]: con partición padre basada en Linux.

- Microsoft Windows Server 2008 R2 con Hyper-V [14]: Con partición padre basada en Windows Server 2008 R2.

Hoy por hoy, estos productos comerciales no pueden competir con las cuotas de mercado que tiene su gran competidor, y referente en el mundo de la virtualización, VMware. Sin embargo, los expertos parecen coincidir en que algo está cambiando en el mercado. Las tecnologías de sus competidores están mejorando mucho y comienzan a reclamar su cuota de mercado. Todo indica que, e adelante, VMware deberá mantenerse alerta si no quiere perder la situación de privilegio que posee actualmente [7] [8].

6. REQUERIMIENTOS

En anteriores apartados se han ido marcando los objetivos y las tecnologías que conformarán el presente proyecto. En este apartado se concretará el entorno de trabajo a utilizar, los actores que forman parte de dichos entornos y los requisitos que determinarán cómo han de interactuar.

Para comenzar, se definirán dos tipos distintos de entorno de trabajo:

- Entorno Físico: Este entorno simulará el que utilizaría el usuario como entorno de trabajo habitual. Estará formado por el Puesto fijo que, para este proyecto, como se menciona en el apartado 3, se compondrá de un PC de sobremesa con sistema operativo Linux.

- Entorno Virtual: Es el que simulará el entorno que portará el usuario cuando salga de viaje. Engloba, tanto el Puesto móvil, como el Puesto virtual y, para este proyecto, estará formado por un PC tipo portátil con los siguientes elementos instalados:

  - Sistema operativo Windows Vista.

  - Hypervisor VMware Server 2.0

  - Una máquina virtual VMware con características similares a las del PC del entorno físico.

Esta nueva clasificación por entornos, aunque no se trata de un requerimiento propiamente dicho, permitirá que las sucesivas explicaciones se refieran a entornos y no a elementos concretos de dichos entornos.

El objetivo principal es crear una abstracción del la complejidad de los elementos del Entorno Virtual, ya que, en lo que respecta a las funcionalidades necesarias para este proyecto, tanto el PC portátil, como el sistema operativo Windows y el Hypervisor WMware Server no son elementos relevantes ya que, lo único que aportan, es la capa de virtualización necesaria para para conseguir que funcione la máquina virtual de dicho entorno.

Por tanto, cuando de ahora en adelante, se haga referencia al Entorno Virtual, en realidad, se hará referencia a la máquina virtual que corre sobre el hypervisor (de características similares a la del Entorno Físico), sin que importe demasiado los componentes sobre los que corre el propio hypervisor. Precisamente, ¡Esta es la magia de la virtualización!

La siguiente ilustración muestra de forma gráfica lo explicado en el párrafo anterior.

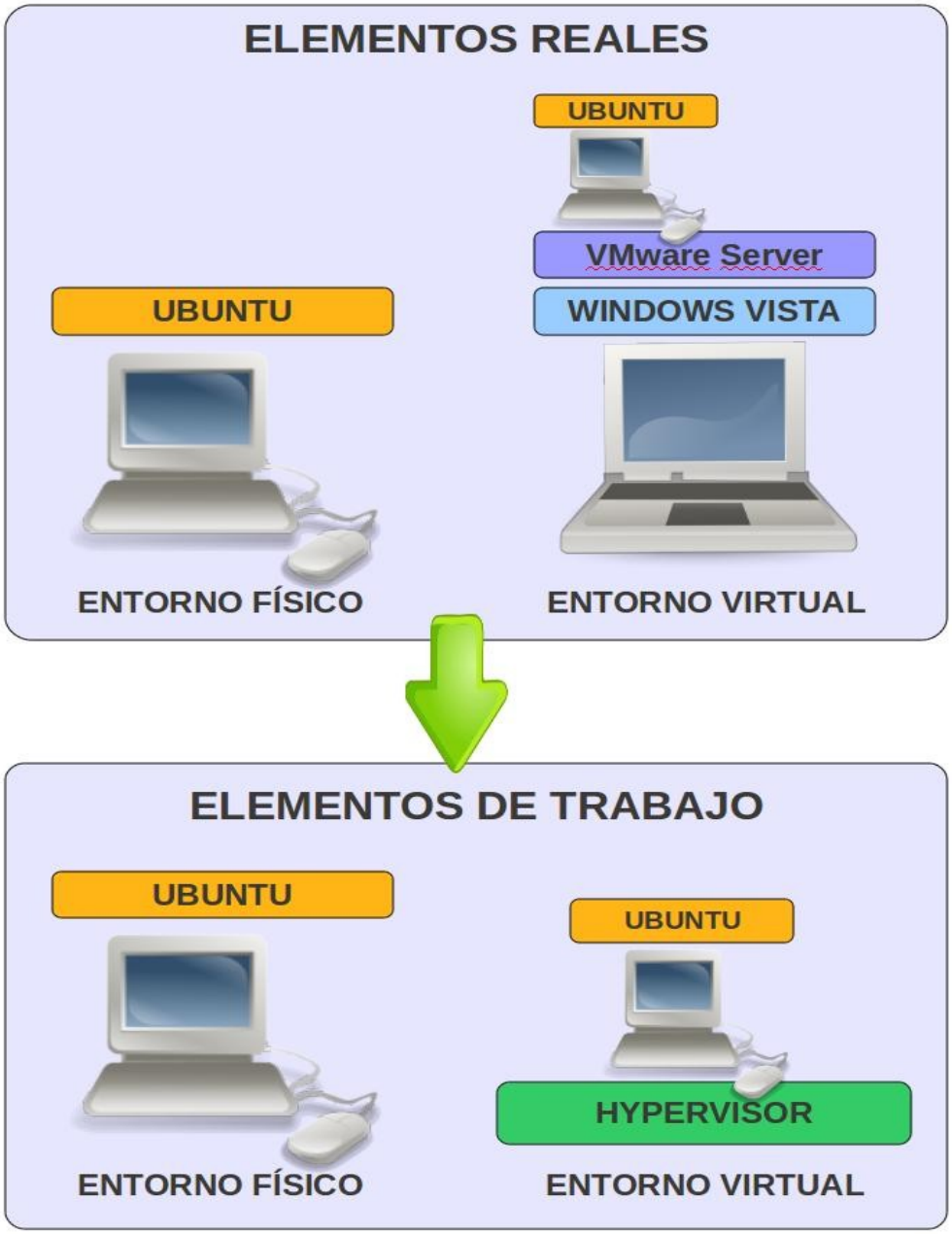

Ilustración 6: Abstracción del Entorno Virtual

Una vez definidas las características de los entornos de trabajo, es preciso también definir los requisitos que deben cumplir:

A) Por un lado, ambos entornos deberán ser similares:

- A nivel hardware bastará con garantizar ciertas similitudes básicas ya que, el Entorno Virtual nunca podrá ser cien por cien idéntico al Entorno Físico. Esto obedece, principalmente, a dos razones:

  - Razones prácticas: Por ejemplo, si el Entorno Físico cuenta con 160 GiB de almacenamiento en disco duro, no es práctico asignar esa

misma cantidad al Entorno Virtual ya que consumirá muchos recursos de la máquina física que soporta el hypervisor y, probablemente no se llegue a necesitar toda esa cantidad de almacenamiento.

- Razones técnicas: Es altamente improbable que el hardware virtual que presenta el hypervisor al sistema operativo de la máquina del Entorno Virtual, coincida con el hardware real del equipo del entorno Físico. Sin embargo, esto no debería ser un problema ya que, si garantizamos un mínimo de compatibilidad (por ejemplo, en la arquitectura del microprocesador), el propio sistema operativo se ocupará de abstraer la capa hardware.

Por las razones vistas anteriormente, será suficiente con exigir que ambos entornos sean de tipo x86 y 32bits. Además, que el Entorno Virtual, en concreto, cuente con la suficiente capacidad de proceso (CPU y RAM), así como almacenamiento suficiente, para ejecutar el sistema operativo y las aplicaciones de uso común del usuario.

- A nivel Software se marcarán unos requisitos más exigentes. Ya que se pretende que el usuario pueda realizar su trabajo, en el Entorno Virtual, de manera parecida a como lo haría en su puesto de trabajo habitual, ambos sistemas operativos se deberán parecer al máximo.

  Por tanto, se exigirá como requisito que ambos sistemas operativos sean los mismos, incluso a nivel de versión. Una vez garantizado este punto, podremos exigir también que los programas usados por el usuario sean también exactamente los mismos e incluso que la estructura de directorios en el filesystem sea, también, lo más parecida posible.

B) Por otro lado, en ambos entornos, se deberá encontrar instalado el script **psync.pl**.

   Dicho script se desarrollará, en lenguaje Perl (v5), como parte del presente proyecto y será el encargado de realizar las tareas de sincronización entre ambos entornos. En apartados posteriores se detallaran las características y funcionalidades de dicho script.

C) Por último, se debe poder contar con algún método para posibilitar el intercambio de ficheros entre los dos entornos.

   Este requisito es fácil de implementar. Para hacerlo se ha optado, en este proyecto, por el uso de un dispositivo tipo “memoria USB” ya que se trata de dispositivos rápidos, asequibles y, hoy en día, cualquier equipo dispone de conector adecuado (incluidas las máquinas virtuales que “reciben” el puerto gracias a que el hypervisor les muestra el puerto físico de las máquinas donde corren).

Sin embargo, esta no es la única opción posible y se podría haber optado por soluciones como una conexión LAN (cada vez más presentes en domicilios particulares), Internet (correo electrónico, FTP,...), etc.

6.1. Requisitos funcionales

A partir de los requisitos vistos en el apartado anterior, se ha delimitado las características del entorno de trabajo del proyecto. En este apartado, se detallarán las funcionalidades de las que dispondrá el usuario cunado interactúe con dicho entorno de trabajo.

El usuario deberá poder realizar las siguientes tres acciones:

- Crear lista de directorios a exportar: Con esta acción, el usuario podrá generar un fichero que contenga una lista de los directorios que desea exportar. Para componer esta lista, se utilizarán dos fuentes distintas:
  - Un fichero de entrada, opcional, en el que el usuario habrá podido añadir, manualmente, una lista de los directorios que él haya considerado necesarios.
  - Uno o varios algoritmos automáticos capaces de obtener directorios en función del uso que el usuario hace del equipo. Los algoritmos deberán ser capaces de analizar el trabajo realizado por el usuario y sugerir los directorios que han sido más utilizados. El objetivo es facilitar al usuario la creación de la lista de directorios evitando olvidos o dependencias no conocidas.

    Como solución concreta, para este proyecto, se implementarán cuatro algoritmos que servirán de ejemplo de la funcionalidad que se quiere obtener en este punto. Los algoritmos a implementar se basarán en:
    - El directorio de trabajo: El algoritmo deberá ser capaz de capturar la ruta del directorio de trabajo del usuario.
    - El fichero History de la sesión de usuario: Este fichero contiene un registro de todos los comandos tecleados por el usuario. El algoritmo, deberá analizar dicho registro e intentar localizar las rutas con las que suele trabajar el usuario.
    - La dependencia de librerías compartidas: A partir de un fichero ejecutable, que indique el usuario, se localizarán todas las librerías compartidas de las que depende dicho ejecutable. El algoritmo deberá recoger las rutas a dichas librerías.

- Dependencias durante el proceso de compilación: Se lanzará un proceso de compilación indicado por el usuario y, durante el proceso, el algoritmo deberá interceptar todas las llamadas a los distintos ficheros que intervienen en el proceso de compilación para poder sugerir un conjunto de rutas adecuadas que englobe a todos ellos.

Estos dos algoritmos no son los únicos posibles. De hecho, el código del script se presenta de tal forma que sería fácil implementar nuevos algoritmos si se desease.

En cualquier caso, estos algoritmos sólo podrán sugerir directorios y será el usuario quien deberá decidir, en última instancia, si la lista de directorios sugerida le es válida o no.

A continuación se muestra un esquema de casos de uso para esta acción.

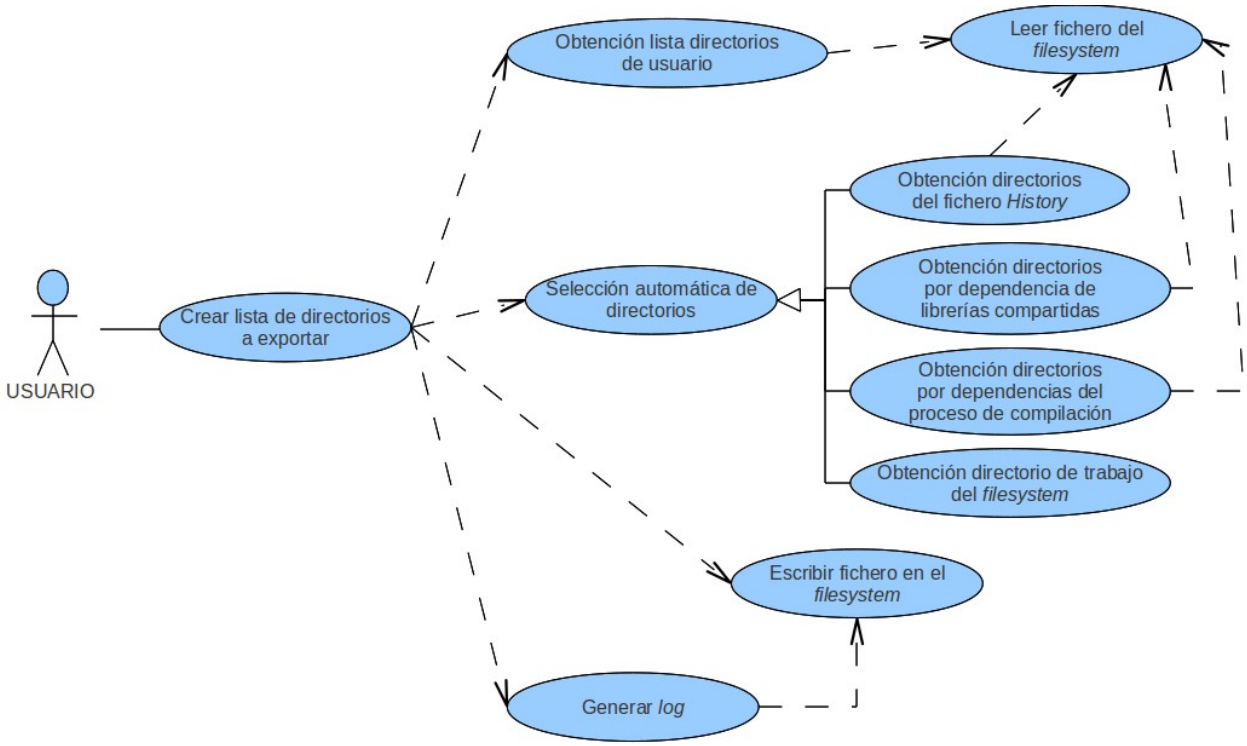

Ilustración 7: Casos de uso: Crear lista de directorios a exportar

- Exportar una lista de directorios a un fichero: Esta acción leerá un fichero con una lista de directorios, obtenido mediante la acción anterior, y obtendrá una copia del contenido de cada uno de dichos directorios.

  El conjunto de datos conseguidos mediante este proceso se escribirá de forma “empaquetada” en un único fichero de salida. Este fichero de salida deberá tener un formato adecuado para que, más adelante, sea posible invertir el proceso, es decir, que sea posible “desempaquetar” los datos y volver a obtener los directorios y ficheros originales.

A continuación se muestra el correspondiente esquema de casos de uso.

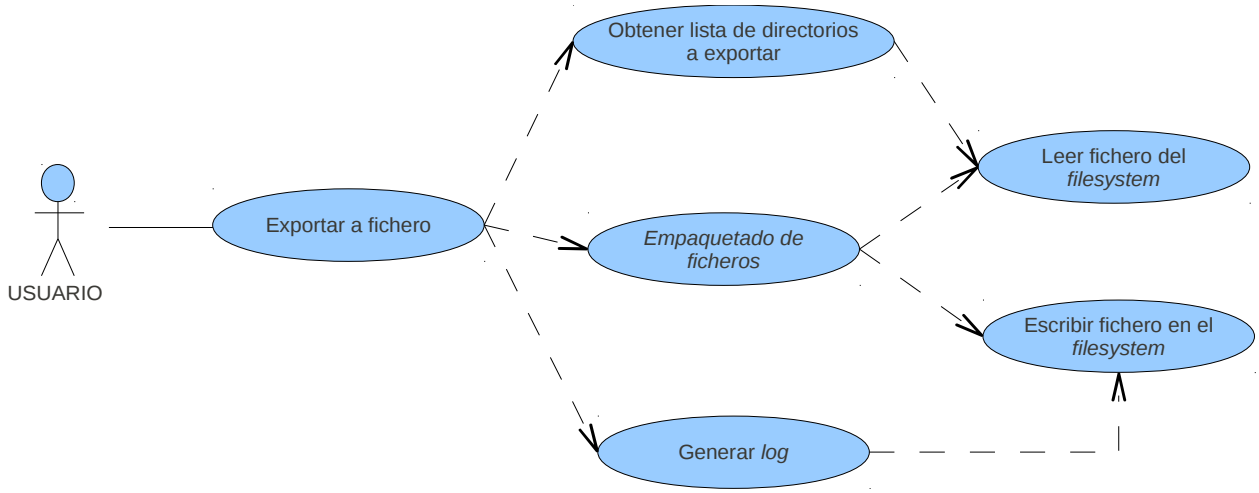

Ilustración 8: Casos de uso: Exportar a fichero

- Importar un fichero escribiendo su contenido en el *filesystem* local: Esta acción, como ya se ha apuntado en la explicación anterior, consistiría en el procedimiento, propiamente dicho, de “desempaquetar” un fichero de entrada, “empaquetado” mediante la acción anteriormente vista, y crear en el *filesystem* local los directorios y ficheros obtenidos.

  Sin embargo, en este punto, se añade un requisito importante ya que, lo que se desea, es sincronizar y no simplemente copiar. Por tanto, en el procedimiento de esta acción se deberá implementar un algoritmo que tenga en cuenta lo siguiente:

  - Si los directorios y ficheros a crear no existen previamente en el filesystem local, se crearán sin más.

  - Si existen directorios y ficheros con el mismo nombre (*path* completo). Se verificará la fecha y sólo se sobreescribirán si los ficheros a escribir tienen fecha de modificación posterior a los ya existentes.

  Una vez más se muestra un esquema de casos de uso para representar la presente acción.

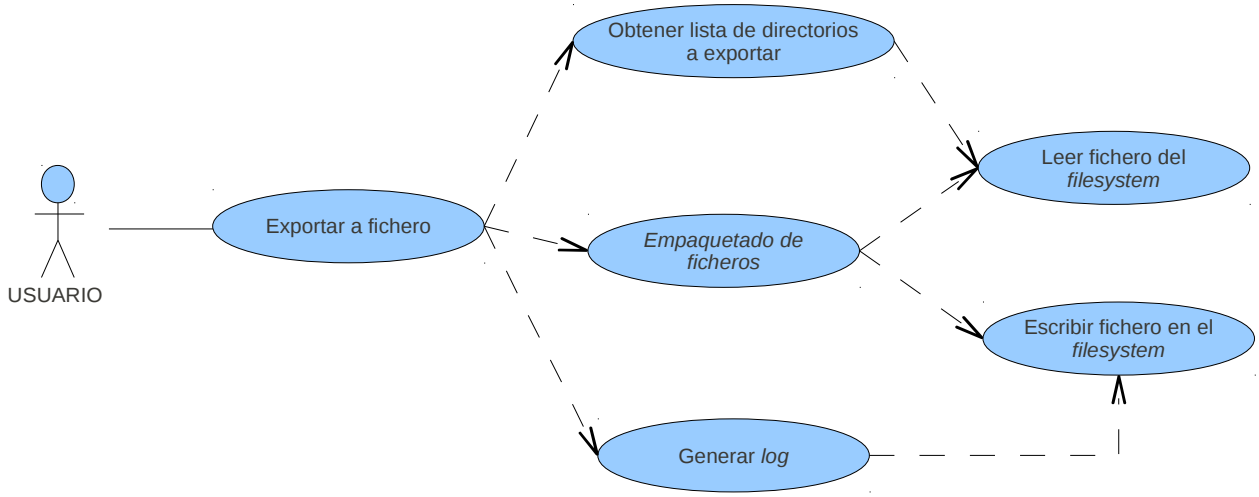

Ilustración 9: Casos de uso: Exportar a fichero

Para terminar, como se puede apreciar en los distintos esquemas de casos de uso, cada una de estas tres acciones dispone de una funcionalidad común que consiste en permitir al usuario la posibilidad de activar un *log* detallado de los pasos realizados durante la ejecución de la acción seleccionada. Dicho *log* se escribirá en un fichero de salida.

7. ARQUITECTURA DEL SISTEMA

La arquitectura del sistema que es necesario implementar para este proyecto es intencionadamente sencilla. Ya que se pretende representar una solución al alcance de un hipotético usuario final, los elementos elegidos para formar la citada arquitectura son, siempre, elementos que puedan estar al alcance de este tipo de usuarios.

Algunos de los elementos que componen la arquitectura del proyecto ya se han visto en apartados anteriores. Aún así, a lo largo de este apartado, se repasará un lista completa de los elementos que la componen y la justificación de su elección.

7.1. Entorno Físico

Como ya se ha visto en el apartado 6 (REQUERIMIENTOS), el entorno físico está compuesto por un único PC físico.

Este componente, no requiere de muchas más explicaciones, simplemente pretende simular un puesto de trabajo de un usuario estándar. En este caso, se ha elegido utilizar un PC con arquitectura x86 por ser, sin duda, la más extendida entre los usuarios, tanto domésticos como profesionales, y un sistema operativo Linux Ubuntu (v10.04) [3] que, si bien no puede competir en popularidad con los sistemas operativos de Microsoft, se ha considerado lo suficientemente representativo.

7.2. Entorno Virtual

Este componente ya se ha visto también con anterioridad, por lo que tampoco se entrará en demasiado detalle en este apartado. Baste mencionar que, como en el caso anterior, se ha pretendido representar un soporte móvil estándar que estuviese al alcance de la mayoría de los usuarios. En este sentido, la plataforma física, es decir, PC portátil x86 con sistema operativo Microsoft Windows Vista, es, seguramente, una de las más populares en estos momentos.

Por otro lado, la elección de la plataforma de virtualización, VMware Server, se ha hecho teniendo en cuenta que los productos de WMware cuentan con una sobrada reputación como líderes del mercado, es una plataforma muy conocida entre los usuarios y, además, se distribuye en modalidad *freeware*, es decir, sin coste para el usuario.

Todo lo anterior demuestra que el Entorno Virtual propuesto es también un componente que podría estar al alcance de cualquier usuario estándar.

7.3. Dispositivo intercomunicador

Este componente, con este nombre tan rimbombante, corresponde a un elemento vital en el proyecto ya que se ocupará de que los ficheros generados en el Entorno Físico puedan ser transportados al Entorno Virtual y viceversa.

Para esta imprescindible misión se ha elegido un dispositivo de tipo memoria USB ya que, estos dispositivos, hoy en día, son muy asequibles, fáciles de utilizar, rápidos y permiten almacenar gran cantidad de información. Además, hace tiempo que las memorias USB forman parte de los complementos imprescindibles para cualquier usuario informático. Por lo que son ideales para el presente proyecto.

7.4. Script sincronizador (psync.pl)

Este componente será el encargado de realizar las tareas necesarias para poder llevar a cavo las acciones que se mencionan en el apartado 6.1 (Requisitos Funcionales). Se implementará mediante un *script* escrito en lenguaje Perl (v5.10) [4].

Perl es un lenguaje flexible, de sintaxis similar al C y muy popular entre los programadores, sobre todo, en entornos Linux. De hecho, el interprete de Perl, necesario para poder ejecutar el *script* de este proyecto, se encuentra disponible, entre los paquetes estándar, en la mayoría de las distribuciones Linux más populares y, por supuesto, en la distribución Linux Ubuntu 10.04 que se utiliza en esta arquitectura.

Una vez más, se comprueba que la solución propuesta está al alcance de cualquier usuario ya que, el simple hecho de tener instalado el sistema operativo Ubuntu en sus equipos, les permitirá ejecutar el *script* **psync.pl** sin mayor problema.

Como se ha visto, este *script* es un elemento clave en el presente proyecto y se desarrollará completamente, a medida, para cubrir sus requisitos funcionales. Debido a la importancia de este complemento, se dedicará enteramente el apartado siguiente a explicar su funcionamiento interno.

8. PSYNC.PL; EL SCRIPT QUE SINCRONIZA LOS ENTORNOS

En el apartado 6.1 (Requisitos funcionales) ya se definió una serie de tareas que el usuario debería poder realizar. Por otro lado en el apartado 7 (ARQUITECTURA DEL SISTEMA) se concretó diciendo que dichas tareas se realizarían mediante el uso de un script escrito en lenguaje Perl.

El presente apartado se centrará en el citado *script* que, para este proyecto, se ha llamado **psync.pl**. El los siguientes subapartados se detallará su funcionamiento y las claves más importantes de su funcionamiento.

8.1. Cómo usar psync.pl

Psync.pl es un script escrito en lenguaje Perl (versión 5.10) y diseñado para ser ejecutado desde la línea de comandos de un sistema operativo Linux. Concretamente, durante el desarrollo del presente proyecto, se utilizó en los Linux Ubuntu de los Entornos Físico y Virtual.

La primera línea del *script* es el *shebang* "`!# /usr/bin/perl -w`" lo que permitirá la ejecución directa del *script* con solo activar los atributos de ejecución adecuados del fichero psync.pl.

La sintaxis correcta para realizar una ejecución del *script* es la siguiente:

```
psync.pl PARAMETROS
```

Todos los parámetros se ajustan siempre al mismo formato. Se indicarán mediante una cadena de caracteres que estará compuesta por las siguientes partes:

- Comenzará siempre por dos guiones (`--`).
- Inmediatamente después se encontrará el nombre del parámetro.

  Todos los parámetros dispondrán de un formato largo, más explicativo e intuitivo, y un formato corto, formado por un sólo carácter, más rápido de teclear. Ambos formatos se consideran sinónimos, es decir, a la hora de teclear el nombre del parámetro se podrá elegir entre el formato largo o corto indistintamente (aunque, lógicamente, sólo se podrá utilizar uno de los dos).
- Por último, si el parámetro requiere que el usuario le asigne algún valor, se deberá añadir, justo a continuación del nombre del parámetro, un signo igual (=) e, inmediatamente después, el valor que se quiera asignar.

A continuación se mostrará la lista de parámetros que se pueden utilizar. Por cada parámetro se mostrará primero su nombre largo y, justo debajo, su nombre corto. Después se añadirá una descripción de su utilidad.

`--accion`
`--a`

Es el parámetro más importante ya que define la acción que realizará el script. El parámetro acción es obligatorio (excepto con el uso de `--help`) y, aunque se puede colocar en cualquier posición, se recomienda, por claridad, que se indique siempre en primer lugar.

Sólo se podrá indicar una acción por comando y sólo se le podrá asignar uno de los tres valores siguientes:

- `ldir` (lista de directorios): Crea un fichero de salida con la lista de directorios que se sugiere exportar. Ya que, durante la creación de esta lista pueden haber intervenido algoritmos automáticos, conviene que la lista sea repasada y depurada por el usuario, con el fin de no exportar más directorios de los realmente necesarios.

  Esta acción sólo tiene, como obligatorio, el parámetro `--output`. Al que se le deberá asignar el nombre del fichero (incluida la ruta) en el que se desea almacenar la lista final de directorios.

- `exp` (exportar): Crea un fichero de salida de tipo .tgz (tar [17] comprimido con gzip [18]) que contiene todos los datos necesarios para poder recrear los directorios que se ha solicitado exportar. Este fichero de salida será el que se deberá transportar entre el Entorno Físico y el Entorno Virtual mediante la memoria USB.

  Esta acción tiene dos parámetros obligatorios. Por un lado se debe indicar un fichero de entrada (parámetro `--input`) que contenga la lista de directorios a exportar y que, generalmente, se habrá generado mediante la opción anterior.

  Por otro lado, será necesario indicar también la ubicación y el nombre del fichero de salida donde se almacenarán los datos exportados (parámetro `--output`).

- `imp` (importar): Esta acción recibe un fichero de entrada con datos exportados mediante la acción anterior e importa dichos datos volcándolos en el *filesystem* sobre el que se está ejecutando la acción. Por tanto, si esta acción se ejecuta en un *filesystem* B, a partir de los datos exportados de un *filesystem* A (diferente), las estructuras de directorios, y los ficheros contenidos en ellos, del *filesystem* A quedarán reproducidas en el *filesystem* B.

Sin embargo, la importación no es una simple copia de directorios y ficheros. Para que la sincronización entre entornos pueda realizarse, la importación sólo copia las estructuras (directorios y ficheros) del fichero de exportación que no existan o que sean más recientes (tengan una fecha de actualización mayor), que las existentes en el *filesystem* de destino.

Esta acción sólo tiene un parámetro obligatorio. El parámetro `--input` con el que habrá que indicar la ubicación y el nombre del fichero que contiene los datos a importar.

`--add-cur-dir`
`--c`

Añade el actual directorio de trabajo (en el momento de la ejecución del *script*) a la lista de directorios que se desea exportar.

Este parámetro no requiere de ningún valor, es opcional y sólo se puede utilizar cuando la acción que va a realizar es la de crear la lista de directorios.

`--add-history-dir`
`--t`

Analiza el fichero History (por defecto ubicado en "`~/bash_history`") donde se almacenan los comando tecleados por el usuario e intenta localizar los directorios utilizados por dicho usuario. Los directorios obtenidos como resultado del análisis, son añadidos a la lista de directorios que se desea exportar.

Este parámetro no requiere de ningún valor, es opcional y sólo se puede utilizar cuando la acción que va a realizar es la de crear la lista de directorios.

`--favorites`
`--f`

Permite añadir, a petición del usuario, una serie de directorios a la lista de directorios a exportar. Mediante esta opción se indica un fichero, que previamente habrá sido creado por un usuario, con una serie de directorios que, dicho usuario, desee que figuren obligatoriamente en la lista de directorios a exportar.

Este parámetro requiere indicar la ruta y el nombre del fichero que contiene la lista de directorios a agregar. El parámetro es opcional y sólo se puede utilizar cuando la acción que va a realizar es la de crear la lista de directorios.

```
--help
--h, --?
```

Permite obtener por pantalla un texto de ayuda con la lista completa de parámetros y un resumen de su funcionalidad.

```
--input
--i
```

Permite indicar la ruta y el nombre de un fichero que será tratado como fichero de entrada para la acción a realizar.

Este parámetro requiere indicar la ruta y el nombre del fichero de entrada. El parámetro es obligatorio en las acciones de importar y exportar.

```
--lib-dependences
--d
```

Analiza un fichero ejecutable de tipo ELF (*Executable and Linkable Format*) [25] y, a partir de los datos de su cabecera, obtiene la lista completa de las librerías compartidas de las que depende para su ejecución. Las rutas en las que se ubican las librerías compartidas obtenidas, son añadidas a la lista de directorios que se desea exportar.

Este parámetro requiere indicar la ruta y el nombre del fichero ejecutable que se deberá analizar y sólo se puede utilizar cuando la acción que va a realizar es la de crear la lista de directorios.

```
--log
--l
```

Permite crear un fichero que contenga información detallada de los pasos realizados durante la ejecución de la acción solicitada.

Este parámetro requiere indicar la ruta y el nombre del fichero donde se guardará la información de los pasos realizados. El parámetro es opcional y puede ser utilizado con cualquiera de las acciones.

```
--make-calls
--m
```

Lanza un proceso de compilación mediante el uso de la utilidad GNU Make [19]. Esta utilidad necesita de un fichero (llamado generalmente “makefile”) donde se escriben la instrucciones que permiten llevar a cabo los pasos adecuados para la correcta compilación de uno o varios ficheros fuente.

Esta opción permite además monitorizar todo el proceso de compilación, capturando, durante dicho proceso, todas las llamadas a ficheros externos para

elaborar una lista de dependencias del proceso de compilación. Las rutas implicadas en estas dependencias, son añadidas a la lista de directorios que se desea exportar.

Este parámetro requiere indicar la ruta y el nombre del fichero “makefile” que permite realizar el proceso de compilación. Sólo se puede utilizar cuando la acción que va a realizar es la de crear la lista de directorios.

`--output`
`--o`

Permite indicar la ruta y el nombre de un fichero que será tratado como fichero de salida para la acción a realizar.

Este parámetro requiere indicar la ruta y el nombre del fichero de salida. El parámetro es obligatorio en las acciones de crear lista de directorios y exportar.

Para terminar con este apartado, se presentarán a continuación una serie de ejemplos sobre el uso de los parámetros vistos, así como, de la correcta sintaxis de la llamada al script psync.pl

Ejemplo 1:

```
psync.pl --a=ldir --output=/dir1/lista.txt --add-cur-dir --t
```

En este caso se ha solicitado la creación de una lista de directorios a exportar. La lista se generará en el fichero de salida “`/dir1/lista.txt`” y la lista de directorios estará formada por el actual directorio de trabajo más los encontrados en el análisis del fichero de *history*.

Ejemplo 2:

```
psync.pl --a=ldir --output=/dir1/lista2.txt --m=/dir2/makefile
```

Se ha solicitado, nuevamente, la creación de una lista de directorios a exportar. La lista se generará en el fichero de salida “`/dir1/lista2.txt`” y estará compuesta por aquellos directorios que contengan ficheros a los cuales se haya hecho referencia desde el proceso de compilación orquestado por el fichero “`/dir2/makefile`”.

Ejemplo 3:

```
psync.pl --a=exp --input=/dir1/lista.txt --o=/dir2/dump.tgz
```

En este otro ejemplo se ha solicitado la exportación de la lista de directorios "`/dir1/lista.txt`". El volcado de los datos exportados se realizará sobre el fichero de salida "`/dir2/dump.tgz`".

Ejemplo 4:

```
psync.pl --a=imp --input=/dir2/dump.tgz --log=/dir3/logimp.log
```

En este ejemplo se ha solicitado la importación del fichero de datos "`/dir2/dump.tgz`". Además se creará un fichero "`/dir3/logimp.log`" que contendrá información detallada de los pasos realizados durante la importación.

Ejemplo 5:

```
psync.pl --help
```

En este último ejemplo se ha solicitado el texto de ayuda.

8.2. Detalle del funcionamiento

El listado completo del *script* psync.pl se puede ver en el ANEXO A. En él se puede observar que las tres principales subrutinas del código son las que llevan a cabo cada una de las tres acciones definidas (crear lista de directorios, exportar e importar). El resto del código esta compuesto por una serie de subrutinas auxiliares y, por supuesto, la rutina principal (*main*) que es el punto de entrada al script y la rutina encargada de que todo se ejecute en el orden esperado.

A continuación se hará un repaso por cada una de las citadas partes, explicando en detalle el funcionamiento de cada una de ellas.

8.2.1. Rutina principal (*main*)

En Perl esta rutina no existe como una rutina diferenciada (como, por ejemplo, en Java o en C). Sencillamente el código empieza a ejecutarse por la primera línea que no

pertenece a una subrutina declarada.

En el caso de psync.pl la rutina principal es muy sencilla y no realiza ninguna labor más allá de coordinar las llamadas a las subrutinas que son las que, realmente, realizan el trabajo.

A continuación se muestra un diagrama de flujo que muestra las tareas que se desempeñan en esta rutina principal.

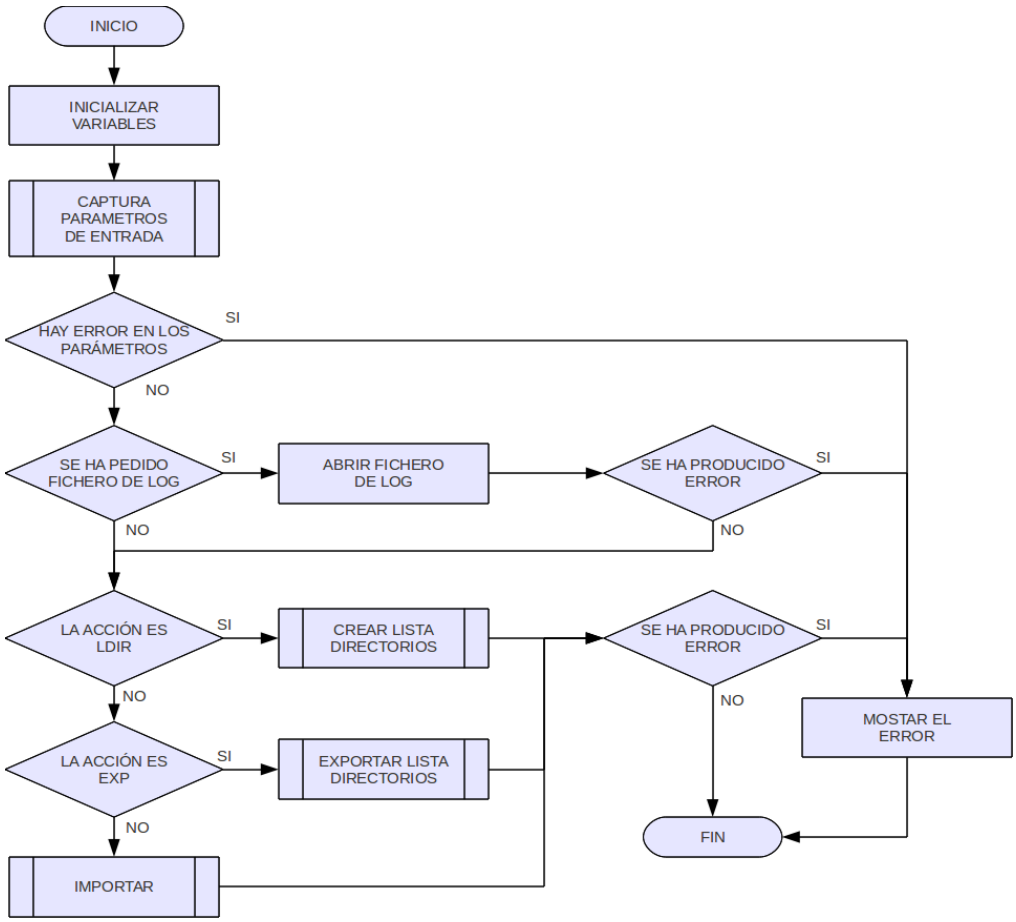

Ilustración 10: Diagrama de flujo de la rutina principal

8.2.2. Rutinas auxiliares

En este apartado se engloban toda una serie de subrutinas de servicio que, siendo útiles para el correcto funcionamiento del script, son sólo piezas de apoyo para las otras rutinas que son las que, realmente, realizan el trabajo.

Entre estas subrutinas de servicio se encuentran la que captura los parámetros del comando de entrada, las que formatean las cabeceras y los pies del fichero de log, la que muestran por pantalla un mensaje con la ayuda del comando, etc.

Todas ellas son sencillas y no necesitan de mayor explicación. Si acaso, la única que merece una mención especial es la subrutina que captura los parámetros de entrada del comando ya que es la más compleja y, además, es la única rutina que hace uso de un módulo de Perl. En este caso se trata del módulo `Getopt::Long` [16].

8.2.3. Rutina para crear la lista de directorios

Esta rutina es una de las más importantes de todo el *script* ya que se ha pretendido que tuviese un cierto grado de “inteligencia”. El objetivo era que el usuario no tuviese que estar pendiente de los directorios que necesitaba exportar para poder seguir trabajando en el Entorno Virtual, si no que, el *script*, fuese capaz de darse cuenta de los directorios que el usuario necesitaría para poder ofrecérselos en una lista que él sólo tuviese que supervisar. Aunque también se contempló la necesidad de que el usuario pudiese incluir una serie de directorios concretos que, sin ser usados frecuentemente, tuviesen información personal que el usuario también desease exportar

Teniendo en cuenta todo esto, esta subrutina tiene un diseño modular. Pensado para poder añadir distintos algoritmos que sean capaces de recopilar los directorios adecuados para el usuario desde distintas fuentes y juntarlos todos en una misma lista de salida única.

Los posibles algoritmos a utilizar son muy variados y pueden depender, incluso, del tipo de usuario al que vayan dirigidos. Así, por ejemplo, para un usuario programador puede ser interesante intentar localizar los últimos ejecutables en los que ha estado trabajando y, después, elaborar una lista con los directorios que mantengan dependencias de compilación con dichos ejecutables. Sin embargo, para un usuario dedicado al diseño gráfico, podría ser interesante acceder a los ficheros de configuración de las herramientas de diseño que utiliza para obtener de allí la lista con los directorios que utiliza como favoritos para dejar sus diseños e incluso acceder a las listas LRU (*Last Recent Used*) de ficheros guardados para saber donde guardó sus últimos diseños.

En cualquier caso, como en los objetivos del presente proyecto se ha marcado la intención de mostrar las virtudes de la virtualización con la vista puesta en un usuario final dedicado al mundo de la informática, que trabaja en entornos Linux, se han elegido una serie de algoritmos basados en utilidades, propias del entorno Linux, que podrían ser útiles para este tipo de usuarios.

En resumen, a la hora de ejecutar la acción “crear una lista de directorios” con el *script* psync.pl, los directorios se pueden obtener de cinco fuentes distintas dependiendo de los parámetros utilizados en la llamada. A continuación se detallarán cada una de estas fuentes:

- Directorio actual de trabajo: Esta es la fuente más sencilla y directa. Simplemente se obtiene del *filesystem* el directorio de trabajo en el momento de la ejecución del *script* y se añade a la lista de directorios a exportar.

Esta fuente no se puede considerar un algoritmo propiamente dicho pero se ha añadido para demostrar la capacidad del *script* para trabajar con múltiples fuentes de directorios. Por otro lado, es obvio que, el directorio en el que el usuario suele trabajar, será seguramente uno de los que desee exportar.

Esta fuente se puede activar añadiendo el parámetro `--add-cur-dir` en la llamada al script.

- Fichero de favoritos: Esta fuente también es bastante directa. Se trata de que el usuario suministre un fichero con una lista de directorios que él desea que se añadan.

  Esta fuente se activa indicando el parámetro `--favorites` en la llamada al *script* y adjuntando la ruta a un fichero con la lista de directorios. El script se limitará a localizar el fichero indicado, abrirlo, y añadir los directorios leídos a la lista de directorios a exportar.

- Fichero *History*: Esta fuente, junto con las dos que se mostrarán a continuación, son un claro ejemplo de lo que sería un algoritmo orientado a la búsqueda proactiva de directorios para sugerir al usuario.

  En este caso, el conocimiento de las tareas realizadas por el usuario se obtiene del fichero *History* (Situado por defecto en `~/.bash_history`) donde se van guardando, a modo de histórico, los últimos comandos que el usuario ha ido tecleando durante las últimas sesiones.

  El objetivo del algoritmo utilizado es analizar los comandos almacenados en dicho histórico e intentar deducir cuales son los directorios visitados por el usuario y en cuales realiza algún tipo de trabajo, es decir, que no sólo visita el directorio, si no que, además, ejecuta comandos en él más allá de un simple "`ls`" o un "`pwd`".

  Entrar en detalle escrito de como actúa este algoritmo es complicado por lo que se adjunta, en la página siguiente, un diagrama de flujo en modo "seudocódigo" que refleja, gráficamente, el comportamiento del algoritmo.

  Esta fuente se puede activar añadiendo el parámetro `--add-history-dir` en la llamada al script.

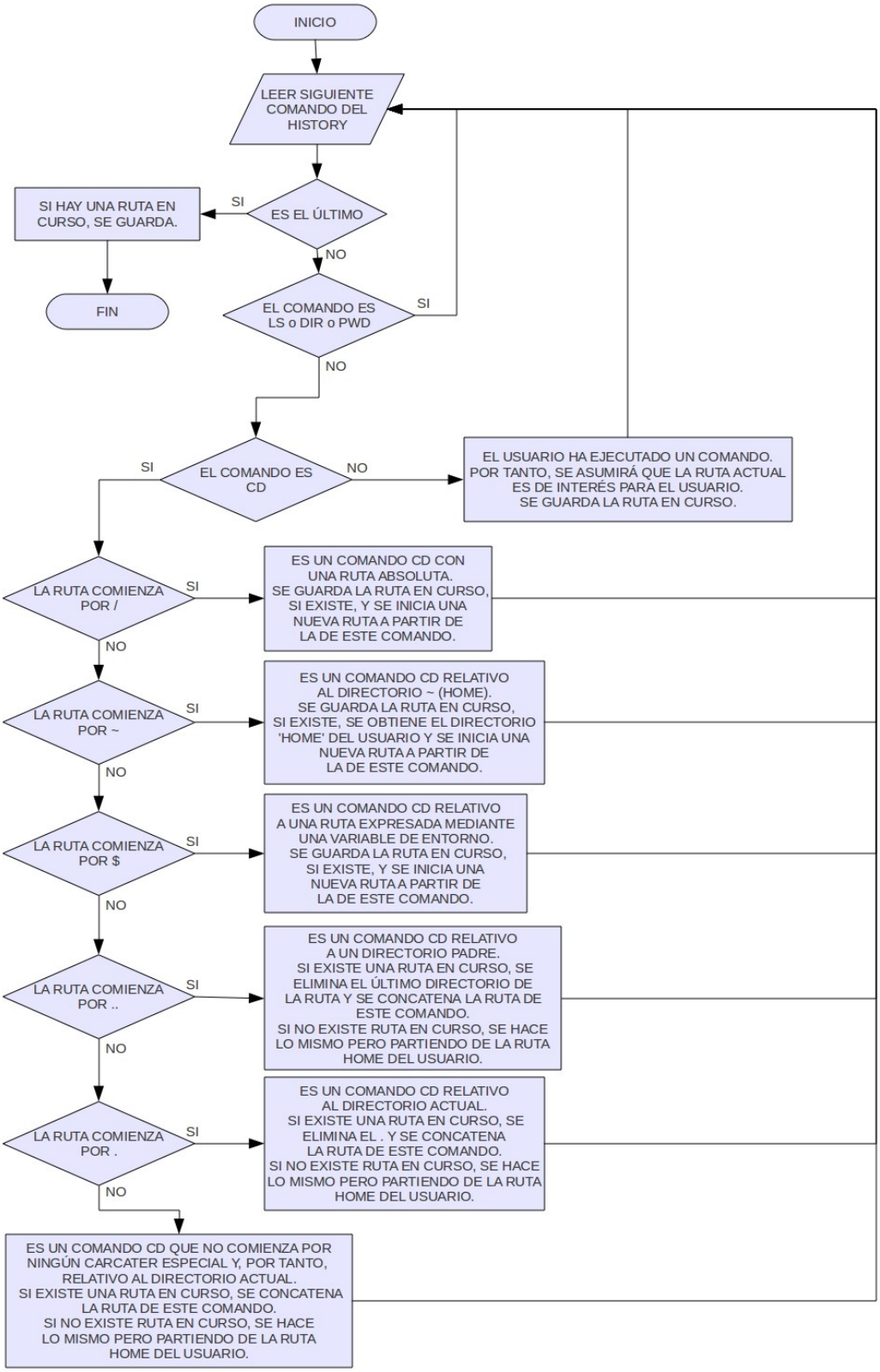

Ilustración 11: Diagrama de flujo del algoritmo que analiza el fichero History.

- Dependencia de librerías compartidas: En este caso, la información de trabajo se obtendrá de los metadatos de un fichero ejecutable del tipo ELF (*Executable and Linkable Format*) [25].

  El objetivo del algoritmo es leer los metadatos del fichero ejecutable mediante la utilidad "`readelf`" (esta utilidad forma parte del paquete de utilidades GNU Binutils [21]) y localizar en ellos la información relativa a las librerías compartidas necesarias para la correcta ejecución del fichero. Teniendo en cuenta, además, que las librerías obtenidas mediante este procedimiento pueden tener, a su vez, dependencias hacía otras librerías. Por lo que ha sido preciso implementar un mecanismo de recursividad que permitiese llegar hasta el último nivel de dependencia. La lista de directorios a sugerir se formará a partir de aquellos directorios que contengan las librerías compartidas necesarias.

  Una peculiaridad de este algoritmo es que necesita que el usuario tenga permiso de `sudo` ya que se utiliza este comando para hacer exploraciones del *filesystem* completo. Por tanto, durante la ejecución de este algoritmo, se solicitará, de forma interactiva por pantalla, la contraseña de `sudo` del usuario.

  Esta fuente se puede activar añadiendo el parámetro `--lib-dependences` en la llamada al script y adjuntando la ruta al fichero ejecutable, con formato ELF, cuyas dependencias se van a analizar.

- Dependencias del proceso de compilación: En este caso, la lista de directorios se obtendrá de las dependencias que se detecten durante una compilación. De esta forma nos aseguraremos de exportar todos los ficheros necesarios para realizar la compilación.

  En esta ocasión el algoritmo debe monitorizar una compilación para obtener sus dependencias. Existen múltiples formas de realizar una compilación, por lo que a sido preciso concretar un método para poder realizar la implementación del algoritmo. Para este proyecto, se ha optado por utilizar un método muy usado en el mundo Linux; la utilidad GNU Make [19].

  Para conseguir compilar un programa mediante la utilidad Make, es preciso construir un fichero, llamado normalmente makefile, donde se describe mediante instrucciones (a modo de script) los pasos necesarios para la correcta compilación de un proyecto.

  Basándose en todo esto, lo que hace el algoritmo es obtener del usuario un fichero makefile y llamar a la utilidad Make pasándole dicho fichero. Esto provoca la ejecución de todos los pasos contenidos en el makefile, obteniéndose, como resultado, la compilación completa del proyecto definido en el makefile.

Al mismo tiempo, todo este proceso de compilación es monitorizado mediante la utilidad "`strace`" (utilidad Linux que permite monitorizar las llamadas al sistema o *system calls* que realiza un determinado proceso en ejecución). De todas las llamadas al sistema que son interceptadas por "`strace`" se filtran aquellas que son de apertura de ficheros (*syscall OPEN*) ya que, éstas, son un claro marcador de todos aquellos ficheros que el proceso a necesitado abrir para completarse con éxito. Una vez capturadas estas llamadas, sólo hay que obtener la información de las rutas a las que se hacían y añadir dichas rutas al fichero de rutas sugeridas para la exportación.

Esta fuente se puede activar añadiendo el parámetro `--make-calls` en la llamada al script y adjuntando la ruta al fichero makefile con el que poder lanzar el proceso de compilación.

Por último, es preciso mencionar que las rutas de los directorios que se obtienen desde cualquiera de las tres fuentes vistas anteriormente, no son presentadas directamente al usuario, si no que, con ellas, se crea una lista completa preliminar que, posteriormente, pasa por un proceso de depuración donde son eliminadas todas las rutas repetidas, no válidas o que apuntan a rutas inexistentes. Una vez pasado el citado proceso de depuración, las rutas resultantes son las que pasarán a formar parte del fichero de salida con la lista de rutas sugeridas para su exportación.

8.2.4. Rutina de exportación

Esta rutina es la encargada de "empaquetar" los directorios, y los ficheros que contienen, en un solo fichero de salida autocontenido, es decir, dicho fichero debe poder ser trasladado a otro Entorno distinto y, sin más ayuda que el propio *script* psync.pl, debe poder ser "desempaquetado" obteniéndose los mismos fichero y estructuras de directorios que se "empaquetaron" en origen.

Para realizar las labores de empaquetado y desempaquetado de información el *script* psync.pl delega en una conocida aplicación de los entornos Linux que, en sus orígenes, ya fue diseñada para realizar este tipo de tareas. Se tarta de la aplicación GNU Tar [17]. Esta aplicación es capaz de empaquetar una serie de ficheros y directorios que se le indique, produciendo un fichero de salida que contiene todos los datos relativos a las estructuras copiadas. Estos datos incluyen información como la fecha de última modificación, que luego será de vital importancia para garantizar la sincronización durante el proceso de importación. Además, la aplicación Tar es capaz de trabajar junto con algunos tipo de compresores (GNU Gzip [18] en el caso de psync.pl) para comprimir el fichero de salida y así conseguir reducir su tamaño, lo que facilita su manipulación.

Por tanto, en lo que respecta al *script* psync.pl, durante la ejecución de la rutina de exportación lo que se hace es verificar que las rutas que se ha solicitado exportar son realmente válidas (es decir, existen realmente) y, si es así, las rutas son pasadas a la aplicación Tar para que ésta genere el fichero de salida.

Cabe destacar que el fichero de exportación generado por la aplicación Tar no se genera directamente donde el usuario lo solicita con el parámetro `--output` si no que se genera en una ubicación temporal dentro del directorio `/tmp`. Esto se hace así para evitar que, si la ubicación del fichero de salida indicada por el usuario está dentro de uno de los directorios a exportar, el fichero de exportación se intente exportar a si mismo.

Una vez finalizado con éxito el proceso de exportación, el fichero resultado es movido a la ubicación solicitada por el usuario y renombrado adecuadamente.

8.2.5. Rutina de importación

Esta rutina es la encargada del “desempaquetado” de todos los datos exportados mediante la rutina anterior. Ya que la rutina anterior empaquetó los datos utilizando la aplicación GNU Tar, ésta también utilizará la misma aplicación para desempaquetarlos.

En este caso, el desempaquetado es algo más delicado ya que no se desea una simple sobrescritura de las estructuras de datos. Lo que se desea es una sincronización para lo cual, el *script* psync.pl, utiliza una funcionalidad incorporada en Tar que evita la sobrescritura de los datos si el fichero que se pretende sobrescribir tiene una fecha de modificación mayor que el que se está importando.

De esta forma, cuando se pase el fichero de exportación del Entorno Virtual al Entorno Físico y se realice la importación, sólo serán reescritos al Entorno Físico aquellos ficheros que hubiesen sido modificados en el Entorno Virtual ya que, sólo estos, tendrán una fecha de modificación posterior.

Por último mencionar que, durante la ejecución de esta rutina, justo antes de lanzarse la llamada a la aplicación Tar, se mueve el fichero de entrada con los datos a importar a una ubicación temporal (en el directorio `/tmp`). En este caso, esta acción se realiza para evitar que, en el caso de que el fichero de datos a importar se encontrase en uno de los directorios que se esta sincronizando, se produjese una sobrescritura de dicho fichero por accidente. Una vez terminada con éxito la importación, el fichero de datos, vuelve a moverse a su ubicación original.

9. PRUEBAS

En este apartado

9.1. Prueba 1

Para esta primera prueba se supondrá que el usuario está trabajando sobre el propio script psync.pl y que desea exportar todo su trabajo, así como la documentación generada y algunos documentos de consulta de programación en Perl.

En definitiva, del Entorno Físico se quiere exportar :

- El directorio de trabajo (`/home/luis/TFG/script`)
- El directorio de documentación (`/home/luis/TFG/documentacion`)
- El directorio con documentos de Perl (`/home/luis/Perl`)

Para resolver esta tarea, en esta ocasión se utilizará la siguiente estrategia:

- Se creará un fichero `favoritos.txt` con el siguiente contenido:

```
/home/luis/TFG/documentacion
/home/luis/Perl
```

- Se lanzará el script psync.pl desde el directorio de trabajo y se pedirá, por parámetro que se añada dicho directorio de trabajo a la lista de directorios a exportar.

Por tanto, para obtener la lista de directorios a exportar, en el Entorno Físico, se utilizará el siguiente comando:

```
psync.pl --accion=ldir --output=./lista-dir.txt --add-cur-dir
      --favorites=./favoritos.txt --log=./log-ldir.log
```

Al ser probado dicho comando en el PC del Entorno Físico del Laboratorio, montado para este proyecto, se obtiene el siguiente resultado:

- Tiempo de ejecución: <1 s.
- Fichero de salida (`/home/luis/TFG/script/lista-dir.txt`)

```
/home/luis/Perl
/home/luis/TFG/documentacion
/home/luis/TFG/script
```

A partir del fichero obtenido, se realiza la exportación mediante el comando:

```
psync.pl --accion=exp --input=./lista-dir.txt
   --output=./exp.tgz --log=./log-exp.log
```

El resultado de la ejecución de dicho comando el en PC del Entorno Físico es el siguiente:

- Tiempo de ejecución: ≈1 s.
- Fichero de salida (`/home/luis/TFG/script/exp.tgz`) con un tamaño de 9947172 bytes (≈9,5 MiB)

El fichero obtenido (`exp.tgz`) y el *script* psync.pl se copian en una memoria USB y se traslada al Entorno Virtual. Ya en el Linux del Entorno Virtual se copian los dos ficheros en el directorio `/home/luis` y, desde allí, se lanza la importación de datos mediante el comando:

```
psync.pl --accion=imp --input=./exp.tgz
          --log=./log-imp.log
```

El resultado de la ejecución de dicho comando en el Entorno Virtual es el siguiente:

- Tiempo de ejecución: <1 s.
- En la máquina virtual se han creado los tres directorios exportados del Entorno Físico y, además, se han generado todos los ficheros que contenía cada uno de los directorios.

Llegados a este punto, esta prueba se da por finalizada habiéndose alcanzado los objetivos.

9.2. Prueba 2

Para la siguiente prueba se continuará con el mismo supuesto anterior, es decir, se exportarán los mismos tres directorios del Entorno Físico pero, esta vez, se incluirán los siguientes cambios:

- Se modificará la estrategia para la obtención de la lista de directorios. En esta ocasión, ya que el usuario trabaja frecuentemente con estos directorios, se intentará obtener dichos directorios analizando el histórico de comandos (fichero History).

- Por otro lado se verificará la capacidad de sincronización del *script*. Para ello, se seleccionará un fichero del PC del Entorno Físico, se verificará el contenido, se exportará, se importará en el Entorno Virtual, se realizará una modificación del fichero en este último entorno, se hará una exportación del Entorno Virtual ((con el fichero ya modificado) y, por último, se importarán los datos en el Entorno Físico. Tras todos estos pasos, el fichero elegido, deberá aparecer, en el Entorno Físico, con las modificaciones aplicadas en el Entorno Virtual.

Lo primero será crear un fichero que pueda ser modificado en el Entono Virtual. Para ello, se crea en el directorio `/home/luis/TFG/documentacion` un fichero mediante el comando.

```
echo Fichero creado en el Entorno Físico > pru-sinc.txt
```

Este comando crea el fichero `pru-sinc.txt` cuyo único contenido es el texto "`Fichero creado en el Entorno Físico`". Este fichero, será el que se modificará, más adelante, en el Entorno Virtual.

El siguiente paso será la obtención de la lista de directorios a exportar mediante el análisis del histórico de comandos (fichero History). Para ello, se lanza el siguiente comando:

```
psync.pl --accion=ldir --output=./lista-dir.txt
     --add-history-dir --log=./log-ldir.log
```

Al ser probado dicho comando en el PC del Entorno Físico del Laboratorio, montado para este proyecto, se obtiene el siguiente resultado:

- Tiempo de ejecución: <1 s.

- Fichero de salida (`/home/luis/TFG/script/lista-dir.txt`)

```
/etc/emacs23
/home/luis
/home/luis/ASO
/home/luis/Perl
```

```
/home/luis/TFG
/home/luis/TFG/documentacion
/home/luis/TFG/script
/tmp
/usr/bin
/usr/games
```

Como se puede observar, los directorios deseados (marcados en negrita) se han podido obtener de entre las entradas del fichero History. Para continuar, se limpiará el fichero `lista-dir.txt`, dejando únicamente los tres directorios a exportar y, después, se realiza la exportación mediante el comando:

```
psync.pl --accion=exp --input=./lista-dir.txt
   --output=./exp2.tgz --log=./log-exp.log
```

El resultado de la ejecución de dicho comando el en PC del Entorno Físico es el siguiente:

- Tiempo de ejecución: ≈1 s.
- Fichero de salida (`/home/luis/TFG/script/exp2.tgz`) con un tamaño de 9948115 bytes (≈9,5 MiB)

El fichero obtenido (`exp2.tgz`) y el *script* psync.pl se copian en una memoria USB y se traslada al Entorno Virtual (Todos los datos copiados o generados durante la Prueba 1 en el Entorno Virtual han sido previamente borrados). Ya en el Linux del Entorno Virtual se copian los dos ficheros en el directorio `/home/luis` y, desde allí, se lanza la importación de datos mediante el comando:

```
psync.pl --accion=imp --input=./exp2.tgz
          --log=./log-imp.log
```

El resultado de la ejecución de dicho comando en el Entorno Virtual es el siguiente:

- Tiempo de ejecución: <1 s.
- En la máquina virtual se han creado los tres directorios exportados del Entorno Físico y, además, se han generado todos los ficheros que contenía cada uno de los directorios.

Una vez importados todos los datos, se localiza el fichero `/home/luis/TFG/documentacion/pru-sinc.txt` en el PC del Entorno Virtual y, con algún tipo de editor, cambiamos la frase que contiene por esta otra "`Fichero`

`modificado en el Entorno Virtual`".

Tras modificar el fichero de referencia, se vuelve a realizar una exportación pero, esta vez, del Entrono Virtual. Para ello, en la máquina virtual Linux se accede al directorio `/home/luis/TFG/script` y, desde allí, se lanza el siguiente comando:

```
psync.pl --accion=exp --input=./lista-dir.txt
   --output=./exp3.tgz --log=./log-exp.log
```

Recordemos que el fichero con la lista de directorios a exportar (`lista-dir.txt`) ya existirá puesto que habrá sido importado en el Entorno Virtual con el resto de ficheros del directorio.

El resultado de la ejecución de dicho comando el en PC del Entorno Virtual es el siguiente:

- Tiempo de ejecución: ≈2 s.
- Fichero de salida (`/home/luis/TFG/script/exp3.tgz`) con un tamaño de 9949155 bytes (≈9,5 MiB)

El fichero obtenido (`exp3.tgz`) se copian nuevamente en la memoria USB y se traslada al Entorno Físico. Una vez copiado el fichero de datos en el directorio `/home/luis/TFG/script` del Entorno Físico, se lanza la importación de datos mediante el comando:

```
psync.pl --accion=imp --input=./exp3.tgz
          --log=./log-imp.log
```

El resultado de la ejecución de dicho comando en el Entorno Físico es el siguiente:

- Tiempo de ejecución: <1 s.
- El contenido del fichero `/home/luis/TFG/documentacion/pru-sinc.txt` es "`Fichero modificado en el Entorno Virtual`".
- En el PC del Entorno Físico no se ha cambiado ningún otro fichero de los importados.

La forma que existe de verificar que ningún otro fichero ha sido modificado es mirar el fichero de *log*, en este caso, la salida del *log* se ha realizado sobre el fichero `./log-imp.log`. Este fichero presenta información detallada sobre la acción realizada. En el caso de la importación, muestra información sobre el resultado de la importación para cada uno de los ficheros. A continuación se muestra un pequeño fragmento del contenido

del fichero `log-imp.log` donde se puede observar que los fichero no han sido copiados al PC del Entorno Físico porque éstos ya existían y sus fecha de modificación son iguales (o más recientes) que las de los ficheros que se pretenden importar.

```
Resultado de la importación:
tar: El actual `/home/luis/Perl/Makefile' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/prueba.c' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/HolaMundo.pl' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/HolaMundo.pl~' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/salidamake.txt' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/prueba.o' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/prueba.exe' es más moderno o de la misma edad
tar: El actual `/home/luis/Perl/manual/perltut.pdf' es más moderno o de la misma
edad
tar: El actual `/home/luis/Perl/Makefile~' es más moderno o de la misma edad
tar: El actual `/home/luis/TFG/documentacion/TFG_2011_memoria_v7.odt' es más
moderno o de la misma edad
tar: El actual `/home/luis/TFG/documentacion/TFG_2011_memoria_v12.pdf' es más
moderno o de la misma edad
tar: El actual
`/home/luis/TFG/documentacion/Informe_Problemas_Desviaciones_PEC2.odt' es más
moderno o de la misma edad
tar: El actual `/home/luis/TFG/documentacion/salida.pdf' es más moderno o de la
misma edad
.....
```

Una vez más, la prueba se da por finalizada habiéndose alcanzado los objetivos.

9.3. Prueba 3

En la prueba anterior se ha podido verificar que los mecanismos de sincronización funcionan adecuadamente, por lo tanto, las siguientes pruebas se centrarán en la exportación de los datos del Entorno Físico y su importación en el Entorno Virtual.

Para esta tercera prueba se supondrá que, el usuario, a parte de querer exportar sus tres directorios de trabajo (los mismos usados en las pruebas anteriores), se querrá asegurar de que el *script* psync.pl funcione adecuadamente en el Entorno Virtual para lo cual decide exportar también todas las dependencias del interprete perl, lo que debería garantizar que éste funcionase correctamente allí donde se importase.

En resumen, del Entorno Físico se exportarán :

- Los directorios:
  - `/home/luis/TFG/script`
  - `/home/luis/TFG/documentacion`
  - `/home/luis/Perl`

- Los directorios que contengan librerías de las cuales dependa la ejecución del interprete de perl (en la distribución de Ubuntu Linux con la que se trabaja en este proyecto el interprete de perl es `/usr/bin/perl`)

Para resolver esta tarea, se utilizará la siguiente estrategia:

- Se creará un fichero `favoritos.txt` con el siguiente contenido:

```
/home/luis/TFG/documentacion
/home/luis/TFG/script
/home/luis/Perl
```

- Se lanzará el script psync.pl y se pedirá, por parámetro, que se busquen las dependencias del ejecutable `/usr/bin/perl`, incluyéndose los directorios adecuados en la lista de directorios a exportar.

Por tanto, para obtener la citada lista de directorios a exportar, en el Entorno Físico, se utilizará el siguiente comando:

```
psync.pl --accion=ldir --output=./lista-dir.txt
        --lib-dependences=/usr/bin/perl
    --favorites=./favoritos.txt --log=./log-ldir.log
```

Al ser probado dicho comando en el PC del Entorno Físico del Laboratorio, montado para este proyecto, se obtiene el siguiente resultado:

- Tiempo de ejecución: 2 m. 33 s.

- Fichero de salida (`/home/luis/TFG/script/lista-dir.txt`)

```
/home/luis/Perl
/home/luis/TFG/documentacion
/home/luis/TFG/script
/lib
/lib/tls/i686/cmov
```

Si se quiere obtener más información sobre las dependencias encontradas se puede mirar el contenido del fichero de *log*, en este caso concreto, se puede ver que las librerías dependientes encontradas son las siguientes:

```
libdl.so.2
libm.so.6
ld-linux.so.2
libc.so.6
libcrypt.so.1
libpythread.so.0
```

En cualquier caso, a partir del fichero de salida obtenido, se realiza la exportación mediante el comando:

```
psync.pl --accion=exp --input=./lista-dir.txt
   --output=./exp4.tgz --log=./log-exp.log
```

El resultado de la ejecución de dicho comando el en PC del Entorno Físico es el siguiente:

- Tiempo de ejecución: 6 m. 26 s.
- Fichero de salida (`/home/luis/TFG/script/exp4.tgz`) con un tamaño de 408766941 bytes (≈390 MiB)

El fichero obtenido (`exp4.tgz`) y el *script* psync.pl se copian en una memoria USB y se traslada al Entorno Virtual (Todos los datos copiados o generados durante las pruebas anteriores, en el Entorno Virtual, han sido previamente borrados). Ya en el Linux del Entorno Virtual se copian los dos ficheros en el directorio `/home/luis` y, desde allí, se lanza la importación de datos mediante el comando:

```
psync.pl --accion=imp --input=./exp4.tgz
          --log=./log-imp.log
```

El resultado de la ejecución de dicho comando en el Entorno Virtual es el siguiente:

- Tiempo de ejecución: 2 m. 45 s.
- En la máquina virtual sólo se han creado los tres directorios “favoritos” exportados del Entorno Físico.

El resultado obtenido, aunque puede sorprender en un principio, es lo que cabría esperar ya que, para este proyecto, se ha partido de un Laboratorio donde, tanto el PC del Entorno Físico, como la máquina virtual del Entrono Virtual, tenían instaladas dos versiones Ubuntu Linux iguales (Ubuntu 10.04). Por tanto, al realizarse la importación en

el Entorno Virtual, se han creado los directorios que no existían (los tres directorios “favoritos”) y no ha sido necesario copiar ninguna de las librerías compartidas ya que, todas ellas, forman parte de los paquetes que componen la distribución Ubuntu montada en ambos entornos.

Una vez más, la prueba se da por finalizada habiéndose alcanzado los objetivos.

9.4. Prueba 4

Para la siguiente prueba, se partirá de un supuesto totalmente distinto. En este caso, se supondrá un usuario que trabaja compilando código y necesita exportar todo lo necesario para seguir compilando su proyecto en el Entorno Virtual.

Para hacer una prueba de cierta complejidad, en lugar de crear unos pequeños ficheros fuente de ejemplo, se utilizará un proyecto de cierta envergadura que se descargará de internet. En concreto se utilizarán los fuentes del proyecto nmap.org [26].

En resumen, el objetivo de la prueba será descargar los fuentes en el Entorno Físico, probar que compilan adecuadamente y, posteriormente realizar una exportación de los datos necesarios para que, una vez importados en el Entorno Virtual, el proyecto siga compilando sin problemas.

Para resolver esta prueba, se utilizará la siguiente estrategia:

- Una vez instalados los fuentes de nmap, se lanzará el script psync.pl y se pedirá, por parámetro, que se reproduzca el proceso de compilación (lanzando el makefile adecuado) para localizar las dependencias de dicho proceso, incluyéndose los directorios necesarios en la lista de directorios a exportar.

Por tanto, el primer paso, es instalar adecuadamente los fuentes del proyecto nmap en el Entorno Físico. Ya que la instalación concreta de los fuentes de nmap no forman parte del ámbito del presente proyecto, baste decir que los fuentes a instalar (`nmap-5.51.tgz`) se descargaron de la página oficial de descargas de nmap.org (http://nmap.org/dowload.html) y que, para su instalación, siguieron también las instrucciones de la web oficial (http://nmap.org/book/inst-source.html).

El proceso final de compilación de la aplicación nmap requiere de la ejecución de dos pasos que, además, deben realizarse en modo superusuario (*root*). Estos dos pasos se deben ejecutar en el directorio donde se han desempaquetado los fuentes son:

- Ejecutar el comando “`make`”: En realidad, esto provoca la ejecución de la utilidad GNU Make utilizando como fichero de instrucciones el fichero `Makefile` que se desempaqueta junto con los fuentes. Esto realiza una serie de tareas generales.

- Ejecutar el comando "`make install`": Una vez más se ejecuta la utilidad GNU Make usando el fichero de instrucciones `Makefile` pero, esta vez, se pide la ejecución de un bloque concreto de instrucciones (esto viene indicado por el parámetro `install`). Esto termina la instalación y permite que el sistema operativo reconozca el nuevo comando (`nmap`).

Los dos pasos mencionados se deben ejecutar en el directorio donde se han desempaquetado los fuentes que, en este caso, es `/home/luis/TFG/nmap-5.51`.

El problema es que, la opción que se ha implementado para hacer un seguimiento de las llamadas OPEN durante la ejecución de make (`--make-calls`) en el *script* psync.pl no está diseñada para hacer múltiples pasos ni para hacer llamadas con parámetros, por tanto, ha sido preciso encontrar una solución a este problema para poder utilizar el *script* psync.pl.

La solución ha sido crear un nuevo fichero llamado `makefileAux`, en el directorio `/home/luis/TFG/nmap-5.51`, cuyo contenido se muestra a continuación.

```
all : paso1

.PHONY : all

paso1 :
	make
	make install
```

La idea es pedir la ejecución de la utilidad GNU Make utilizando, como *script* de compilación, el nuevo fichero `makefileAux`. Esto permitirá que, con una sola llamada a Make y sin parámetros, se puedan realizar los dos pasos necesarios para la instalación de nmap. Falta un último detalle, como se mencionó anteriormente, los dos comandos finales se deben hacer en modo superusuario (*root*) por tanto, antes de lanzar el *script* psync.pl se deberá lanzar un comando "`sudo su`" para escalar privilegios.

Una vez todo preparado, se procederá a obtener la lista de directorios a exportar. Para ello, en el Entrono Físico, se accede al directorio `/home/luis/TFG/script` y se lanzarán los siguientes comandos:

```
sudo su

psync.pl --accion=ldir --output=./lista-dir.txt
--make-calls=/home/luis/TFG/nmap-5.51/makefileAux
--log=./log-ldir.log

exit
```

Al ser probado dicho comando en el PC del Entorno Físico del Laboratorio, montado para este proyecto, se obtiene el siguiente resultado:

- Tiempo de ejecución: 2 m. 38 s.
- Fichero de salida (`/home/luis/TFG/script/lista-dir.txt`)

```
/dev
/etc
/home/luis/TFG/nmap-5.51
/home/luis/TFG/nmap-5.51/zenmap/zenmapCore
/lib
/lib/i686/cmov
/lib/tls/i686/cmov
/proc
/usr/include
/usr/include/arpa
/usr/include/asm
/usr/include/asm-generic
/usr/include/bits
/usr/include/c++/4.4
/usr/include/c++/4.4/backward
/usr/include/c++/4.4/bits
/usr/include/c++/4.4/debug
/usr/include/c++/4.4/ext
/usr/include/c++/4.4/i486-linux-gnu/bits
/usr/include/gnu
/usr/include/linux
/usr/include/linux/can
/usr/include/linux/hdlc
/usr/include/net
/usr/include/netinet
/usr/include/python2.6
/usr/include/rpc
/usr/include/sys
/usr/lib
/usr/lib/gcc/i486-linux-gnu/4.4.3
/usr/lib/gcc/i486-linux-gnu/4.4.3/../../../../lib
/usr/lib/gcc/i486-linux-gnu/4.4.3/include
/usr/lib/gcc/i486-linux-gnu/4.4.3/include-fixed
/usr/lib/gconv
/usr/lib/locale
/usr/lib/locale/es_ES.utf8
/usr/lib/locale/es_ES.utf8/LC_MESSAGES
/usr/lib/python2.6
/usr/lib/python2.6/config
/usr/lib/python2.6/dist-packages
/usr/lib/python2.6/distutils
/usr/lib/python2.6/distutils/command
/usr/lib/python2.6/encodings
/usr/local/bin
/usr/local/lib/python2.6
/usr/local/lib/python2.6/dist-packages
```

```
/usr/local/lib/python2.6/dist-packages/radialnet
/usr/local/lib/python2.6/dist-packages/radialnet/bestwidgets
/usr/local/lib/python2.6/dist-packages/radialnet/core
/usr/local/lib/python2.6/dist-packages/radialnet/gui
/usr/local/lib/python2.6/dist-packages/radialnet/util
/usr/local/lib/python2.6/dist-packages/zenmapCore
/usr/local/lib/python2.6/dist-packages/zenmapGUI
/usr/local/lib/python2.6/dist-packages/zenmapGUI/higwidgets
/usr/local/share
/usr/local/share/applications
/usr/local/share/man/de/man1
/usr/local/share/man/es/man1
/usr/local/share/man/fr/man1
/usr/local/share/man/hr/man1
/usr/local/share/man/hu/man1
/usr/local/share/man/it/man1
/usr/local/share/man/jp/man1
/usr/local/share/man/man1
/usr/local/share/man/pl/man1
/usr/local/share/man/pt_BR/man1
/usr/local/share/man/pt_PT/man1
/usr/local/share/man/ro/man1
/usr/local/share/man/ru/man1
/usr/local/share/man/sk/man1
/usr/local/share/man/zh/man1
/usr/local/share/nmap
/usr/local/share/nmap/nselib
/usr/local/share/nmap/nselib/data
/usr/local/share/nmap/nselib/data/psexec
/usr/local/share/nmap/scripts
/usr/local/share/zenmap
/usr/local/share/zenmap/config
/usr/local/share/zenmap/docs
/usr/local/share/zenmap/locale
/usr/local/share/zenmap/locale/de
/usr/local/share/zenmap/locale/de/LC_MESSAGES
/usr/local/share/zenmap/locale/fr
/usr/local/share/zenmap/locale/fr/LC_MESSAGES
/usr/local/share/zenmap/locale/hr
/usr/local/share/zenmap/locale/hr/LC_MESSAGES
/usr/local/share/zenmap/locale/pt_BR
/usr/local/share/zenmap/locale/pt_BR/LC_MESSAGES
/usr/local/share/zenmap/locale/ru
/usr/local/share/zenmap/locale/ru/LC_MESSAGES
/usr/local/share/zenmap/misc
/usr/local/share/zenmap/pixmaps
/usr/local/share/zenmap/pixmaps/radialnet
/usr/share/locale
/usr/share/locale-langpack/es/LC_MESSAGES
/usr/share/locale/es/LC_MESSAGES
```

Lo primero que hay que hacer es verificar si la aplicación nmap ha quedado realmente instalada. Para ellos se lanza la aplicación para obtener alguna funcionalidad sencilla, por ejemplo, mostrar la versión del producto. Al hacerlo se obtiene lo siguiente:

```
nmap --version

Nmap version 5.51 ( http://nmap.org )
```

Con esto se comprueba que la instalación ha sido correcta.

Lo segundo es observar la lista de directorios propuesta. Se deberá decidir si se acepta, tal cual, o se realiza alguna modificación sobre ella.

En el caso concreto de esta prueba, se ha decidido hacer cambios ya que, entre las ocho primeras entradas de la lista, se propone la exportación de las rutas `/dev`, `/etc` y `/proc`. Estos directorios contienen datos de uso interno del sistema operativo y puede ser bastante delicado intentar importarlos en el Entorno Virtual. Por tanto, se acude nuevamente al fichero de *log* (`log-ldir.log`) donde se podrá hacer una búsqueda para descubrir cuales han sido las llamadas realizadas a esos directorios concretos. La búsqueda da como resultado que sólo se han detectado las siguientes llamadas a los citados tres directorios:

```
/dev/null
/dev/tty
/etc/ld.so.cache
/proc/filesystem
/proc/meminfo
```

Este tipo de llamadas parecen demostrar que son llamadas de servicio y que no son tareas de la instalación propiamente dicha. Por tanto, se ignorarán esos tres directorios.

Por otro lado, debido a que el *script* psync.pl siempre utiliza la recursividad a la hora de exportar los directorios, se puede realizar una limpieza de la lista de directorios propuesta, dejándola, finalmente, como sigue:

```
/home/luis/TFG/nmap-5.51
/lib
/usr/include
/usr/lib
/usr/local/bin
/usr/local/lib/python2.6
/usr/local/share
/usr/share/locale
/usr/share/locale-langpack/es/LC_MESSAGES
```

Una vez realizados los procesos de depuración de la lista de directorios, se realiza la exportación mediante el comando:

```
psync.pl --accion=exp --input=./lista-dir.txt
   --output=./exp5.tgz --log=./log-exp.log
```

El resultado de la ejecución de dicho comando el en PC del Entorno Físico es el siguiente:

- Tiempo de ejecución: 14 m. 34 s.
- Fichero de salida (`/home/luis/TFG/script/exp5.tgz`) con un tamaño de 1341272338 bytes (≈1,2 GiB)

El fichero obtenido (`exp5.tgz`) se copian en una memoria USB y se traslada al Entorno Virtual (Esta vez se mantienen los datos de la última prueba por lo que, el *script* psync.pl ya se encuentra copiado en la ruta `/home/luis`). Ya en el Linux del Entorno Virtual se copia el ficheros en el directorio `/home/luis` y, desde allí, se lanza la importación de datos mediante el comando:

```
psync.pl --accion=imp --input=./exp4.tgz
          --log=./log-imp.log
```

El resultado de la ejecución de dicho comando en el Entorno Virtual es el siguiente:

- Tiempo de ejecución: 13 m. 7 s.
- Cuando se escribe el comando nmap en una consola del PC del Entrono Virtual, se devuelve un error indicando que la aplicación nmap no está instalada.

Para intentar determinar por qué ha fallado la prueba se revisa el fichero de *log* (`log-imp.log`) en busca de pistas. Buscando la palabra “nmap”, se descubre que no se ha podido crear el directorio `/usr/local/share/namp` por un fallo de permisos. Es posible que si, durante la instalación, se necesitaron permisos de superusuario (*root*), también se necesiten durante la importación.

Para comprobarlo, se vuelve a lanzar el mismo comando de importación pero precedido por un “`sudo su`” para escalar privilegios. El resultado es aún peor ya que el sistema se vuelve inestable y, tras un reinicio, el PC del Entorno Virtual no vuelve a arrancar.

Para terminar se decide hacer una última prueba. Ésta consistirá en volver a repetir la exportación de los ficheros del Entorno Físico pero, esta vez, realizándola también en modo superusuario (*root*). Tras el correspondiente escalado de privilegios (con "`sudo su`") en el entorno Físico se procede a lanzar el mismo comando de exportación visto anteriormente. En esta ocasión, el resultado es el siguiente:

- Tiempo de ejecución: 17 m. 4 s.
- Fichero de salida (`/home/luis/TFG/script/exp5.tgz`) con un tamaño de 1672681284 bytes (≈1,6 GiB)

Una vez más, se traslada el fichero `exp5.tgz` al PC del entorno virtual y se realiza la importación en modo superusuario (*root*). El resultado es el siguiente:

- Tiempo de ejecución: 15 m. 39 s.
- Esta vez, el sistema parece estable. Las aplicaciones responden con normalidad y, tras un reinicio, el sistema arranca con normalidad.
- Cuando se escribe el comando nmap en una consola del PC del Entrono Virtual, no ocurre nada. Es decir, esta vez no se obtiene ningún error indicando que la aplicación nmap no está instalada pero, la aplicación, tampoco responde.

Por tanto, la prueba se da por finalizada sin haber alcanzado los objetivos. Si bien hay que decir que el *script* psync.pl no se diseño con el objetivo de realizar este tipo instalaciones "móviles".

Por otro lado, hay que mencionar que, gracias a la virtualización, la recuperación del PC del Entorno Virtual se realizó en poco más de siete minutos ya que, su recuperación, sólo supuso sobreescribir los ficheros que utiliza el hypervisor (en el PC portátil) por los que se guardaron como copia de seguridad cuando se creo el Laboratorio.

10. CONCLUSIONES

El objetivo del presente proyecto a sido poner de manifiesto la versatilidad de la tecnología de la virtualización. Una tecnología que, como se ha podido ver a lo largo del presente trabajo, se puede aplicar en entornos tan distintos como grandes empresas con gran cantidad de servidores en sus CPD, usuarios profesionales o simples usuarios domésticos.

Como se ha visto en los primeros apartados, los beneficios para las grandes empresas, con numerosos activos, son casi evidentes y ya se ocupan las grandes campañas de marketing, organizadas por las compañías desarrolladoras de las tecnologías de virtualización, de recordarnos esas grandes cifras. Sin embargo, este proyecto ha querido centrarse en buscarle, a la tecnología de virtualización, una utilidad para los usuarios de “a pie”. Sin olvidar que, algo que puede ser útil a un usuario doméstico, puede ser extrapolado, a mayor escala, a usuarios profesionales y entonces, la utilidad, comienza a traducirse en beneficios económicos.

Por tanto, a lo largo del presente proyecto se ha elaborado una hipótesis de trabajo, con un usuario “viajero”, y se ha ido desarrollando una solución que cubría las necesidades de dicho usuario. La mencionada solución se ha creado a partir de la premisa de la virtualización pero sin perder de vista que se estaba desarrollando una solución para un hipotético usuario particular, por lo que, se ha utilizado, en todo momento, tecnología sencilla, al alcance de cualquier usuario doméstico y, sobre todo, económica. De hecho, salvo en lo referente al hardware (que se asume que ya poseía el usuario), el coste de la solución propuesta en este proyecto es de cero euros.

Durante los sucesivos apartados de la presente memoria se ha ido desgranando el desarrollo completo del proyecto. Se ha definido la hipótesis de trabajo, se han extraído los requisitos de la solución, se ha definido una arquitectura adecuada a dichos requisitos y, por último, se ha implementado la solución. Dicha implementación ha supuesto la configuración de dos entornos de trabajo (un Entorno Físico y otro Virtual) y la creación de un *script* (psync.pl) capaz de sincronizar los entornos mencionados anteriormente. En opinión del autor, se ha implementado una solución que ha cubierto completamente las expectativas del proyecto ofreciendo, por un lado, una solución completa a la hipótesis de trabajo y, por otro, poniendo de manifiesto el gran potencial, en cuanto a utilidad y versatilidad, de la tecnología de virtualización.

10.1. Siguientes pasos a dar

Para la realización de este proyecto ha sido necesario acotar la hipótesis de trabajo con el objeto de poder realizar un entregable en un tiempo razonable, sin embargo, las posibilidades que se presentan a partir de la arquitectura de la solución propuesta son muy variadas, sobre todo si se eliminan las restricciones en cuanto a las capacidades de los usuarios finales.

A continuación se presentarán distintas propuestas de mejora sobre la solución inicial propuesta clasificadas según el área en el que se interviene:

- Mejoras en el *script* de sincronización: Como ya se avanzó en el apartado 8.2 (Detalle del funcionamiento [del *script* psync.pl]), el *script* de sincronización se ha diseñado con una intencionada modularidad que permitiera adaptarlo a las necesidades de trabajo del usuario final.

  En ese sentido, la parte del *script* que más se puede adaptar a la forma de trabajo del usuario final es la de la elaboración de la lista de directorios a exportar. La idea es que se utilicen algoritmos de búsqueda de directorios que puedan crear listas lo más cercanas posibles a la necesidades reales del usuario.

  En el presente proyecto se ha presentado una serie de algoritmos, unos con carácter generalista (como el de captura del directorio de trabajo y el fichero *History*) y otros más orientados a trabajos de desarrollo informático (como el de dependencia de librerías y dependencias de compilación), que pretenden ser una muestra de las posibilidades en este aspecto. Está claro que la especialización de este tipo de algoritmos, en éstas y otras áreas, podría dar un importante valor añadido al *script*.

  La dificultad de estos algoritmos radica en que deben ser muy especializados ya que serán tanto más eficaces cuanto mejor sepan qué tipo de información buscan y cual es el mejor sitio para localizarla. En este sentido, por ejemplo, en el presente proyecto se ha utilizado un algoritmo que utiliza la herramienta GNU Make [19] que es una utilidad, muy concreta, utilizada para labores de compilación y que, posiblemente, nunca usará un usuario que no se dedique a dichas tareas.

  En esta línea, una posible mejora a presentar para usuarios con perfil de desarrollador informático sería, por ejemplo, la creación de algoritmos capaces de interactuar con otras herramientas de ayuda a la compilación, como Apache Ant [20].

- Mejoras en la seguridad: Como ya se ha explicado, utilizando el *script* psync.pl para la exportación de los directorios, se obtiene un fichero de salida de tipo .tgz con todos los datos “empaquetados”.

  El problema es que, el formato tgz, es un formato estándar bien conocido (formato Tar [17] comprimido con Gzip [18]) y, por tanto, un hipotético atacante que robase este fichero tgz, no tendría ningún problema en hacerse con la información que contiene en pocos minutos. En el caso de un usuario doméstico quizás esto no tenga demasiada importancia pero, si se comienza a hablar de datos profesionales, la cosa cambia.

  De todos es conocido que la seguridad total no existe, sin embargo, en este caso, sería claramente mejorable si se aplicase algún tipo de cifrado al

fichero de exportación una vez creado.

- Mejoras en la capa de virtualización: En este proyecto se ha utilizado un software de virtualización muy conocido y que está al alcance de cualquier usuario por ser de uso gratuito. Sin embargo, la tecnología de virtualización ofrece múltiples soluciones que se ajustan a muchas otras situaciones.

  Por ejemplo, la solución del proyecto se podría mejorar radicalmente si el presupuesto no fuese un problema (en este caso estaríamos hablando de usuarios profesionales) y se pudiesen aplicar soluciones como las siguientes:

  - Virtualización del puesto cliente o DV (*Desktop virtualization*): Mediante esta solución de virtualización se puede hacer llegar un puesto de trabajo virtual completo hasta el terminal del usuario.

    Con esta tecnología el usuario no dispone de un Entorno Físico tal y como se ha definido para este proyecto. El puesto de trabajo sería en realidad una máquina virtual que se ejecutaría en algún servidor de algún CPD que ni siquiera tiene por que ser local.

    Cuando el usuario desea trabajar en el PC de su escritorio, en realidad, lo que hace, es conectarse a un sistema gestor de virtualización de puesto cliente o VDI (*Virtual Desktop Infraestructure*) que será capaz de asignar un espacio en los servidores del CPD para arrancar una máquina virtual y hacerle llegar al PC del usuario la interfaz de dicha máquina virtual. Por lo tanto, cuando el usuario interactúa con su PC de sobremesa, en realidad, está interactuando, a través de la interfaz que se le presenta, con la máquina virtual que está corriendo en el CPD. Esto consigue independizar al usuario del hardware que realmente soporta su puesto de trabajo. De esta forma, si el usuario necesitase marcharse de viaje sólo necesitaría conectarse, una vez en su destino, con su portátil al mismo sistema VDI. Aunque en esta ocasión utilizaría internet (probablemente mediante el uso de tráfico seguro), el proceso sería el mismo descrito anteriormente y el sistema VDI enviaría la interfaz del puesto de trabajo al portátil del usuario, quien trabajaría exactamente en el mismo ordenador que trabaja normalmente, aunque éste, se encontrase a muchos kilómetros de distancia.

    Soluciones comerciales de este tipo son, por ejemplo, Citrix XenDesktop [22] y VMware View [23].

  - Citrix GoToMyPC [24]: Esta solución de virtualización permite que cualquier usuario, que tenga su PC de trabajo conectado a internet, pueda acceder a él en remoto. De esta forma un usuario puede trabajar directamente en su puesto de trabajo aunque se encuentre de viaje a muchos kilómetros.

Esta solución es, por tanto, otra solución de DV pero, su gran ventaja, es que no necesita de una costosa y complicada VDI ya que la conexión se realiza “punto a punto” entre el usuario y su puesto de trabajo.

11.BIBLIOGRAFÍA

[1] VMware, Inc. *History of Virtualization* [en línea]. <http://www.wmware.com/virtualization/history.html> [Consultada: 12/10/2011]

[2] BITTMAN, Tom. *Server Virtualization: From Virtual Machines To Clouds* [en línea]. Gartner Webinar, 20/01/2010. Disponible en web: <http://www.gartner.com/it/content/1260600/1260616/january_20_virtualization_tbittman.pdf> [Consultada: 14/10/2011]

[3] Página principal de Ubuntu <http://www.ubuntu.com>

[4] Página principal del lenguaje de programación Perl <http://www.perl.org>

[5] Documentación oficial de Perl [en línea]. <http://perldoc.perl.org>

[6] TULLOCH, Mitch. *Understanding Microsoft Virtualization solutions* [en línea]. Microsoft Press, 2010. Disponible en web: <http://download.microsoft.com/download/5/B/4/5B46A838-67BB-4F7C-92CB-EABCA285DFDD/693821ebook.pdf> [Consultada: 3/12/2011]

[7] LOHR, Steve. *Challenging Microsoft With a New Technology* [en línea]. The New York Times, 30/08/2009. Disponible en web: <http://www.nytimes.com/2009/08/31/technology/business-computing/31virtual.html?pagewanted=1&_r=2&partner=rss&emc=rss>

[8] Virtualization.info staff. *VMware ESX still leads the market, XenServer grows almost 300%, Hyper-V grows over 200% says IDC* [en línea]. Virtualization.info, 04/05/2010. Disponible en web: <http://virtualization.info/en/news/2010/05/vmware-esx-still-leads-the-market-xenserver-grows-almost-300-hyper-v-grow-over-200-says-idc.html>

[9] Documentación oficial VMware Server. <http://www.vmware.com/products/server/overview.html>

[10] Documentación oficial VMware Workstation. <http://www.vmware.com/products/workstation/overview.html>

[11] Documentación oficial Microsoft Virtual Server 2005 R2. <http://www.microsoft.com/windowsserversystem/virtualserver/default.aspx>

[12] Documentación oficial VMware vSphere ESXi. <http://www.vmware.com/products/vsphere-hypervisor/overview.html>

[13] Documentación oficial Citrix XenServer. <http://www.citrix.com/English/ps2/products/product.asp?contentID=683148&ntref=prod_top>

[14] Documentación oficial Microsoft Windows Server 2008 R2 Hyper-V. <http://www.microsoft.com/en-us/server-cloud/windows-server/hyper-v.aspx>

[15] VMware Inc. *Comparing VMware vSphere Hypervisor and Windows Server 2008 with Hyper-V* [en línea]. <http://www.vmware.com/products/vsphere-hypervisor/compare.html>

[16] Getopt::Long. Documentación oficial de Perl [en línea]. <http://perldoc.perl.org/5.10.1/Getopt/Long.html>

[17] Página oficial de GNU Tar. <http://www.gnu.org/software/tar/>

[18] Página oficial de GNU Gzip. <http://www.gnu.org/software/gzip/>

[19] Página oficial de GNU Make. <http://www.gnu.org/software/make/>

[20] Página oficial del proyecto Apache Ant. <http://ant.apache.org/>

[21] Página oficial de GNU Binutils. <http://www.gnu.org/software/binutils/>

[22] Documentación oficial Citrix XenDesktop [en línea]. <http://www.citrix.com/virtualization/desktop/xendesktop.html>

[23] Documentación oficial VMware View [en línea]. <http://www.vmware.com/products/view/overview.html>

[24] Documentación oficial Citrix GoToMyPC [en línea]. <http://www.citrix.com/English/ps2/products/product.asp?contentID=13994&ntref=prod_top>

[25] Colaboradores de Wikipedia. Executable and Linkable Format [en línea]. Wikipedia, La enciclopedia libre, 15/09/2011. Disponible en web: <http://es.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=49786030> [Consultada: 7/01/2012]

[26] Página oficial del proyecto nmap.org. <http://nmap.org/>

ANEXO A

A continuación se muestra el contenido completo del *script* psync.pl.

```
#! /usr/bin/perl -w

use strict;
use Getopt::Long;

#----------- Variables golbales -------------

use vars qw(%param $error $flaglog);

#---------- Funciones --------------------

sub dameFecha {
      my ($segundo, $minuto, $hora, $dia, $mes, $anio) = (localtime(time()))
[0,1,2,3,4,5];
      $anio += 1900;   #Los anios se dan a partir del 1900
      $mes += 1;       #Los meses se dan del 0 al 11.
      return sprintf("%02d/%02d/%04d - %02d:%02d:%02d", $dia, $mes, $anio, $hora,
$minuto, $segundo);
}

sub muestraAyuda {
      print "\n\n";
      print "La sintaxis correcta es la siguiente:\n\n";
      print "       psync.pl PARAMETROS\n\n";
      print "PARAMETROS:\n\n";
      print "  --a, --accion=(ldir|exp|imp)\n";
      print "      La acción es siempre obligatoria. ";
      print "Las acciones válidas son:\n";
      print "         ldir : crea un fichero con la lista de los directorios a
sincronizar.\n";
      print "         exp  : crea un fichero con los datos que se desean exportar.\n";
      print "         imp  : importa los datos desde un fichero.\n\n";
      print "  --c, --add-cur-dir\n";
      print "      Añade el directorio de trabajo actual a la lista de directorios\n";
      print "      a exportar. Sólo aplica con --accion=ldir\n\n";
      print "  --d, --lib-dependences=<fichero_ejecutable>\n";
      print "      Descubre las librerías de las que depende <fichero_ejecutable>\n";
      print "      y añade sus rutas a la lista de directorios a exportar.\n";
      print "      Sólo aplica con --accion=ldir\n\n";
      print "  --f, --favorites=<fichero_dir_favoritos>\n";
      print "      Fichero que contendrá la lista de directorios que se desea\n";
      print "      añadir a la exportación. Sólo aplica con --accion=ldir\n\n";
      print "  --h, --?, --help\n";
      print "      Muestra este mensaje de ayuda.\n\n";
      print "  --i, --input=<fichero_entrada>\n";
      print "      Fichero del que se leerán los datos.\n\n";
      print "  --l, --log=<fichero_log>\n";
      print "      Fichero en el que se escribirá el log detallado del proceso.\n\n";
      print "  --m, --make-calls=<fichero_makefile>\n";
      print "      Intercepta las llamadas OPEN durante la ejecución del\n";
      print "      <fichero_makefile> y añade las rutas a la lista de directorios\n";
      print "      a exportar. Sólo aplica con --accion=ldir\n\n";
      print "  --o, --output=<fichero_salida>\n";
      print "      Fichero en el que se escribirán los datos.\n\n";
      print "  --t, --add-history-dir\n";
      print "      Examina el fichero de history (\$HOME/.bash_history) e intenta
elaborar\n";
      print "      una lista de los directorios más usados por el usuario.\n";
      print "      Sólo aplica con --accion=ldir\n\n";
```

```
	print "Para cada acción, los parámetros válidos son los siguientes:\n\n";
	print "  psync.pl --a=ldir --o=<f_sal> [--f=<f_fav>] [--c] [--t]\n";
	print "               [--d=<f_exec>] [--m=<f_mkfile>] [--l=<f_log>]\n\n";
	print "  psync.pl --a=exp --i=<f_ent> --o=<f_sal> [--l=<f_log>]\n\n";
	print "  psync.pl --a=imp --i=<f_ent> [--l=<f_log>]\n\n";
	print "  psync.pl --help\n\n";
}

sub cabeceraLog {
	print LOG
"********************************************************************\n";
	print LOG "* script: psync.pl
*\n";
	print LOG "*
*\n";
	print LOG "* Ejecutado: " . dameFecha . "
*\n";
	print LOG
"********************************************************************\n\n";
	print LOG "Llamada al script psync.pl con los siguientes parametros:\n";
	if ($param{'accion'} eq 'ldir'){
		print LOG "Acción: \"ldir\"\n";
		print LOG "output: $param{'output'}\n";
		print LOG "favotires: $param{'favorites'}\n"
			if (defined $param{'favorites'});
		print LOG "lib-dependences: $param{'lib-dependences'}\n"
			if (defined $param{'lib-dependences'});
		print LOG "make-calls: $param{'make-calls'}\n"
			if (defined $param{'make-calls'});
		print LOG "add-cur-dir: ACTIVADO\n"
			if (defined $param{'add-cur-dir'});
		print LOG "add-history-dir: ACTIVADO\n"
			if (defined $param{'add-history-dir'});
		print LOG "log: $param{'log'}\n" if ($flaglog);
	} elsif ($param{'accion'} eq 'exp'){
		print LOG "Acción: \"exp\"\n";
		print LOG "input: $param{'input'}\n";
		print LOG "output: $param{'output'}\n";
		print LOG "log: $param{'log'}\n" if ($flaglog);
	} elsif ($param{'accion'} eq 'imp'){
		print LOG "Acción: \"imp\"\n";
		print LOG "input: $param{'input'}\n";
		print LOG "log: $param{'log'}\n" if ($flaglog);
	}
	print LOG "\n";
}

sub pieLog {
	print LOG
"********************************************************************\n";
	print LOG "* Terminado: " . dameFecha . "
*\n";
	print LOG
"********************************************************************\n";
}

sub printlog ($) {
	 print LOG $_[0] if $flaglog;
}

sub recogeParam {
	if (! GetOptions (\%param, 'accion|a=s', 'input|i=s', 'output|o=s', 'log|l=s',
			'favorites|f=s', 'add-cur-dir|c', 'add-history-dir|t',
```

```
                    'lib-dependences|d=s', 'make-calls|m=s', 'help|h|?')) {

              print "Error en los parámetros\n";
              $error = 1;
              return;
       }

       if (defined $param{'help'}) {
              #Se ha pedido ayuda.
              $error = 10;
              return;
       }

       if (! defined $param{'accion'}) {
              print "Es obligatorio especificar una acción.\n";
              $error = 1;
              return;
       }

       if ($param{'accion'} eq 'ldir') {
              if (! defined $param{'output'}) {
                     print "El parámetro output es obligatorio ";
                     print "cuando la acción es \"ldir\".\n";
                     $error = 1;
                     return;
              }
       } elsif ($param{'accion'} eq 'exp') {
              if (! defined $param{'input'}) {
                     print "El parámetro input es obligatorio ";
                     print "cuando la acción es \"exp\".\n";
                     $error = 1;
                     return;
              }

              if (! defined $param{'output'}) {
                     print "El parámetro output es obligatorio ";
                     print "cuando la acción es \"exp\".\n";
                     $error = 1;
                     return;
              }
       } elsif ($param{'accion'} eq 'imp') {
              if (! defined $param{'input'}) {
                     print "El parámetro input es obligatorio ";
                     print "cuando la acción es \"imp\".\n";
                     $error = 1;
                     return;
              }
       } else {
              print "La acción \"$param{'accion'}\" no es una ";
              print "acción permitida\n";
              $error = 1;
              return;
       }

       $flaglog = 1 if ($param{'log'});
}

sub historyDir {
       printlog "Se accederá al fichero History ya que se ha ";
       printlog "activado el flag --add-history-dir\n";
       #Se ha pedido crear la lista de directorios a partir del History

       my $dhome = $ENV{'HOME'};
```

```
my $dhist = $ENV{'HISTFILE'};
$dhist = "${dhome}/.bash_history" if ! defined $dhist;
my @listarutas;

if (open HIST, "<$dhist") {
	#fichero History encontrado
	printlog "Encontrado fichero History en $dhist\n";

	my $ruta = "";
	my $aux = "";
	my $pos;

	while (<HIST>) {
		chop;
		next if /(^ls|^dir|^pwd)(.*)/;
		if (/(^cd\s+)(.+)/) {
			#Se ha encontrado un comando que empieza
			#por 'cd' y, por tanto, puede contener
			#una ruta valida.

			# se inicia la concatenacion de la ruta del
			# comando a la nueva direccion
			$aux = $2;
			if ($aux =~ /^\//) {
				# La ruta del comando empieza por '/'
				# por tanto, el comando contiene una
				# ruta absoluta
				if ($ruta ne "") {
					#Si existe una ruta en
					#construccion, se termina
					#para iniciar una nueva.
					push @listarutas, $ruta;
				}
				$ruta = "cd $aux";
			} elsif ($aux =~ /^~(|\/(.+))/) {
				# La ruta del comando empieza por '~'
				# por tanto, el comando contiene una
				# ruta relativa a /home
				if ($ruta ne "") {
					#Si existe una ruta en
					#construccion, se termina
					#para iniciar una nueva.
					push @listarutas, $ruta;
				}
				$ruta = "cd ${dhome}$1";
			} elsif ($aux =~ /^\$(\w+)(|\/(.+))/) {
				# La ruta del comando empieza por ($env)
				# el comando contiene una ruta relativa
				# a una variable de entorno
				if ($ruta ne "") {
					#Si existe una ruta en
					#construccion, se termina
					#para iniciar una nueva.
					push @listarutas, $ruta;
				}
				$ruta = 'cd ' . $ENV{$1};
				if ($2 ne "") {
					$ruta = $ruta . $2;
				}
			} elsif ($aux =~ /^\.\.(|\/(.+))/) {
				# La ruta del comando empieza por (..)
				# (dos puntos) el comando contiene una
				# ruta relativa a un directorio padre
```

```
                        if ($ruta eq "") {
                                #La ruta no esta iniciada,
                                #por tanto, se asume que se
                                #parte de la ruta $HOME
                                $ruta = "cd $dhome";
                        }
                        #Se quita un directorio a la ruta y
                        #se concatena el resto
                        $pos = rindex ($ruta, '/');
                        if ($pos > 0) {
                                $ruta = substr $ruta, 0, $pos;
                        }
                        if ($1 ne "") {
                                $ruta = $ruta . $1;
                        }
                } elsif ($aux =~ /^\.(|\/(.+))/) {
                        # La ruta del comando empieza por (.)
                        # (un punto) el comando contiene una
                        # ruta relativa al directorio actual
                        if ($ruta eq "") {
                                #La ruta no esta iniciada,
                                #por tanto, se asume que se
                                #parte de la ruta $HOME
                                $ruta = "cd $dhome";
                        }
                        #Se quita el punto y
                        #se concatena el resto
                        $ruta .= $1;
                } else {
                        # La ruta del comando no empieza por
                        # ningun caracter especial. Se asumira
                        # que es una ruta normal relativa al
                        # directorio de trabajo actual
                        if ($ruta eq "") {
                                #La ruta no esta iniciada,
                                #por tanto, se asume que se
                                #parte de la ruta $HOME
                                $ruta = "cd $dhome";
                        }
                        $ruta = $ruta . (($ruta =~ /\/$/)?'':'/') . $aux;
                }
        } else {
                #el comando leido no es un 'cd '
                #por tanto, se termina la ruta
                if ($ruta ne "") {
                        #guardar ruta
                        push @listarutas, $ruta;
                        $ruta = "";
                }
        }
}
#Ya no hay mas registros pero debemos verificar
#si no nos dejamos una ruta ya iniciada
if ($ruta ne "") {
        #guardar ruta
        push @listarutas, $ruta;
}

#Se elimina el prefijo 'cd ' en las
#rutas obtenidas del History.
@listarutas = map {$1 if (/^cd\s*(.*)/)} @listarutas;

# Ya tenemos una lista inicial de rutas aunque
```

```
		# muchas de ellas seguro que no son validas.
		# más adelante, se depurará la lista completa.
		printlog "\nSe han obtenido, inicialmente, las siguientes ";
		printlog scalar(@listarutas);
		printlog " rutas del fichero History:\n";
		foreach my $reg (@listarutas) {
			printlog "$reg \n";
		}
		printlog "\n";
	} else {
		#No se ha encontrado el fichero de History
		$error = 9;
	}
	return @listarutas;
}

sub favoritos {
	printlog "Se accederá al fichero de favoritos ya que se ha ";
	printlog "indicado --favorites\n";
	#Se ha pedido agragar una lista de directorios favoritos

	my @listarutas;

	# Se intenta abrir el fichero de favoritos
	if (-e "$param{'favorites'}") {
		# Se intenta obtener los directorios favoritos
		printlog "Encontrado fichero de directorios ";
		printlog "favoritos en $param{'favorites'}\n";

		if (open FAV, "<$param{'favorites'}") {
			# se lee el fichero de favoritos y los
			# directorios se añaden a la lista de rutas
			printlog "Se han incorporado ";
			printlog "los siguientes directorios:\n";

			while (<FAV>) {
				printlog $_;
				chop;
				push @listarutas, $_;
			}

			close FAV;
			printlog "\n";
		} else {
			# No se puede acceder al fichero de favoritos
			$error = 4;
		}
	} else {
		# no existe el fichero de favoritos
		$error = 4;
	}

	return @listarutas;
}

sub libDependences {
	printlog "Se obtendran las librerías dependientes ya que ";
	printlog "se ha indicado --lib-dependences\n";
	# Se ha pedido crear una lista de directorios a partir
	# de las ubicaciones de las librerías de las que
	# depende un ejecutable

	my @listarutas;
```

```
# Se verifica que existe el ejecutable
if (-e $param{'lib-dependences'}) {
	#fichero encontrado
	printlog "Encontrado fichero ejecutable ";
	printlog "$param{'lib-dependences'}\n\n";

	my %res;
	my %temp;
	my $todavia_hay_ceros = 0; #false
	my $ccmd = "";

	# Se lanza un readelf del ejecutable para
	# obtener las dependencias de primer nivel
	$ccmd = "readelf -d $param{'lib-dependences'} | grep \"(NEEDED)\"";
	printlog "Lanzando: $ccmd\n";
	%res = map {($1 => 0) if /.*\[(.+)\]/ } `$ccmd`;
	($todavia_hay_ceros = 1) if %res; #true si %res
				  #tiene algun
				  #elemento

	# Se inicia un bucle que obtendrá las
	# dependencias al resto de niveles
	while ($todavia_hay_ceros) {
		$todavia_hay_ceros = 0; #false para iniciar
				    #un nuevo loop
		%temp = %res;
		while ((my $key, my $value) = each %temp) {
			if ($value == 0){
				$ccmd = "sudo find / | grep --word-regexp ";
				$ccmd .= "--regexp=$key";
				printlog "Lanzando: $ccmd\n";
				my @listaaux = `$ccmd`;
				$res{$key} = 1; #true si ya
						#se ha buscado
						#su ruta
				chop (@listaaux);
				my $ruta = $listaaux[0]; # Se guarda
							 # una ruta a
							 # la librería
				foreach (@listaaux) {
					# Se quita el nombre de la
					# librería de la ruta
					my $pos = rindex ($_, '/');
					if ($pos > 0) {
						$_ = substr $_ , 0, $pos;
						push @listarutas, $_;
					}
				}

				$ccmd = "readelf -d $ruta | grep \"(NEEDED)\"";
				printlog "Lanzando: $ccmd\n";
				@listaaux = map {$1 if /.*\[(.+)\]/ } `$ccmd`;
				foreach (@listaaux) {
					if (!exists $res{$_}){
						$res{$_} = 0;
						$todavia_hay_ceros = 1; #true
					}
				}
			}
		}
	}
```

```
		if ($flaglog) {
			printlog "\nLibrerías dependientes:\n";
			while ((my $key, my $value) = each %res) {
				printlog "$key\n";
			}
			printlog "\nSe han obtenido, inicialmente, las siguientes ";
			printlog scalar(@listarutas);
			printlog " rutas:\n";
			@listarutas = sort (@listarutas);
			foreach (@listarutas) {
				printlog "$_\n";
			}
			printlog "\n";
		}
	} else {
		#No se ha encontrado el ejecutable
		$error = 11;
	}
	return @listarutas;
}

sub makeCalls {
	printlog "Se obtendran las rutas de las llamadas del ";
	printlog "make ya que se ha indicado --make-calls\n";
	# Se ha pedido crear una lista de directorios a partir
	# de las llamadas al syscall OPEN dentro de la
	# ejecución de un make

	my @listarutas;

	# Se verifica que existe el makefile
	if (-e $param{'make-calls'}) {
		#makefile encontrado
		printlog "Encontrado fichero makefile ";
		printlog "en $param{'make-calls'}\n\n";

		# Normalmente la ejecución del make se
		# realiza desde el mismo directorio
		# donde se encuentra el makefile
		# por tanto se cambiará la unicación
		# del directorio de trabajo actual
		my $rutaorigen;
		my $rutadestino;
		my $makefile;
		my $pos;

		my @pru = split (/\//, $param{'make-calls'});
		$pos = @pru;

		if ( $pos < 2 || $pru[0] ne '') {
			print "ERROR, la ruta ha de ser absoluta\n";
		} else {
			#la ruta parece correcta
			$rutaorigen = `pwd`;
			chop($rutaorigen);
			$rutadestino = join('/', @pru[0..($pos-2)]);
			$makefile = $pru[$pos-1];
		}

		printlog "\nDatos obtenidos:\n";
		printlog "Ruta origen = $rutaorigen\n";
		printlog "Ruta destino = $rutadestino\n";
		printlog "Nombre makefile = $makefile\n";
```

```
            chdir $rutadestino if ($rutaorigen ne $rutadestino);

            my @res;
            my $ccmd = "strace -f -e trace=open make --file=$makefile 2>&1 ";
            $ccmd .= "| grep open | grep -v ENOENT";

            printlog "Lanzando: $ccmd\n";
            printlog "Desde la ruta: $rutadestino\n";
            # Se lanza el comando y se desachan todas las
            # líneas de salida excepto las que contienen
            # la info de la llamada al syscall OPEN
            @res = map {$3 if(/(.*)(open\(\")(.*)(\",.*)/)} `$ccmd`;

            # Se eliminan todas aquellas rutas que hacen referencia a
            # ficheros temporales o ficheros situados en la raiz
            # del arbol de direcotirios
            @listarutas = map {$1 if(/(.*)(\/)(.+)/)}
                              grep {!/^((\/tmp)|[^\/])/} @res;

            push @listarutas, $rutadestino;

            if ($flaglog) {
                  printlog "\nLlamadas detectadas:\n";
                  foreach (@res) {
                        printlog "$_\n";
                  }

                  printlog "\nSe han obtenido, inicialmente, las siguientes ";
                  printlog scalar(@listarutas);
                  printlog " rutas:\n";
                  @listarutas = sort @listarutas;
                  foreach (@listarutas) {
                        printlog "$_\n";
                  }
                  printlog "\n";
            }


            chdir $rutaorigen if ($rutaorigen ne $rutadestino);
      } else {
            #makefile no encontrado
            $error = 12;
      }
      return @listarutas;
}

sub creaListaDir{
      # Se ha elegido la accion crear lista de directorios
      my @listatemp;
      my @listadef;
      my $ruta = "";
      my $aux = "";

      if ($param{'add-history-dir'}) {
            push @listatemp, historyDir;
            return if ($error != 0);
      }

      if (defined $param{'favorites'}) {
            push @listatemp, favoritos;
            return if ($error != 0);
```

```
	}

	if (defined $param{'lib-dependences'}) {
		push @listatemp, libDependences;
		return if ($error != 0);

	}

	if (defined $param{'make-calls'}) {
		push @listatemp, makeCalls;
		return if ($error != 0);

	}

	if ($param{'add-cur-dir'}) {
		printlog "Se agregará la ruta del dir. de trabajo ya que ";
		printlog "se ha activado --add-cur-dir\n";
		# Se ha pedido incorporar el directorio de trabajo actual.
		$aux = `pwd`;
		chop($aux);
		printlog "Se añade el directorio local: $aux\n\n";
		push @listatemp, "$aux";
	}

	# Ya tenemos una lista inicial de rutas pero, antes
	# de darla por buena, se limpiara de elementos repetidos
	# y rutas erroneas.
	printlog "Se va a iniciar el proceso de depurado de la lista de rutas
inicial.\n";

	#Se elimina el '/' del final de la ruta, si lo tiene.
	@listatemp = map {(/(.*)\/$/) ? $1 : $_} @listatemp;

	@listatemp = sort @listatemp;
	$aux = "";
	foreach (@listatemp) {
		next if /(^\s*$|^\s*\/\s*$)/;   #Se elimina todas las rutas del tipo
					#'cd' y 'cd /' si no se haría un
					#export de todo el disco.
		if ($_ ne $aux) {
			$aux = $_;
			push @listadef, $_ if (-d);  #Si la ruta no existe,
						#se desecha.
		}
	}

	printlog "Tras la depuración se han obtenido ";
	printlog scalar(@listadef);
	printlog " rutas.\n";

	if (length @listadef > 0) {
		# Si al final del proceso queda alguna ruta valida,
		# se guarda en el fichero de salida.
		if (open SALIDA, ">$param{'output'}") {
			printlog "\nCreando fichero de salida en $param{'output'}.\n";
			printlog "Se están agregando las siguientes rutas:\n";
			foreach (@listadef) {
				print SALIDA "$_\n";
				printlog "$_\n";
			}
			close SALIDA;
		} else {
			# No se ha podido crear el fichero de salida
```

```
                    $error = 3;
             }
      }
}

sub exportar{
      # Se ha elegido la accion exportar
      # Se intenta abrir el fichero de directorios
      if (-e $param{'input'}) {
            printlog "Encontrado fichero de directorios en: $param{'input'}\n";
            my $noencontrado = 0;
            # Se intenta obtener la lista de directorios
            if (open ENTRADA, "<$param{'input'}") {
                  #todo correcto
                  # se lee el fichero de directorios si todos los directorios
                  # a exportar existen.
                  print "\nVerificando directorios a exportar:\n";
                  while (<ENTRADA>) {
                        chop;
                        print "Verificando directorio: $_ ";
                        printlog "Verificando directorio: $_ ";
                        if (-d) {
                              print "-> Existe.\n";
                              printlog "-> Existe.\n";
                        } else {
                              print "-> NO existe.\n";
                              printlog "-> NO existe.\n";
                              $noencontrado = 1;
                        }
                  }

                  close ENTRADA;
                  print "\n";
                  printlog "\n";

                  if ($noencontrado == 0) {
                        #Todas las rutas a exportar existen

                        #Se crea un nombre de fichero temporal. Se utilizara este
fichero
                        #para realizar el export y asi evitar que el fichero de
salida
                        #se intente exportar a si mismo.
                        my $ftmp = '/tmp/f4545_' . $$ . '.tgz';

                        # Se lanza el comando tar
                        my $ccmd = "tar --create --verbose --recursion --absolute-
names ";
                        $ccmd .= "--format=gnu --gzip --file=$ftmp --dereference ";
                        $ccmd .= "--hard-dereference --files-from=$param{'input'}";
                        printlog "Lanzando: $ccmd\n";
                        my $res = 0;
                        $res = `$ccmd`;
                        printlog "Ficheros exportados:\n$res\n";

                        #Verificamos que se ha generado el fichero de export
correctamente.
                        if (-e $ftmp) {
                              printlog "Encontrado fichero de export en: $ftmp\n";

                              # Por ultimo, colocamos el fichero de salida en su
sitio
                              $ccmd = "mv $ftmp $param{'output'}";
```

```
                                printlog "Lanzando el comando: $ccmd\n";
                                $res = `$ccmd`;
                        } else {
                                #No se ha generado el fichero de salida.
                                #algo ha fallado con TAR.
                                $error = 7;
                        }
                } else {
                        #Error: hay alguna ruta a exportar que no existe
                        $error = 6;
                }

        } else {
                # No se ha podido abrir el fichero de directorios
                $error = 5;
        }
    } else {
        # No se ha podido localizar el fichero de directorios
        $error = 5;
    }
}

sub importar{
    # Se ha elegido la accion importar
    # Se intenta localizar el fichero a importar
    if (-e $param{'input'}) {
        printlog "Encontrado fichero de datos en: $param{'input'}\n";

        #Se crea un nombre de fichero temporal. Se utilizara este fichero
        #para realizar el import y asi evitar durante la extraccion de datos
        #se sobrescriba el fichero origen por error.
        my $ftmp = '/tmp/f5454_' . $$ . '.tgz';

        #Se mueve el fichero de datos a /tmp
        my $ccmd = "mv $param{'input'} $ftmp";
        printlog "Lanzando: $ccmd\n";
        my $res = `$ccmd`;

        # Se lanza el comando tar
        $ccmd = "tar --get --verbose --absolute-names ";
        $ccmd .= "--delay-directory-restore --keep-newer-files ";
        $ccmd .= "--dereference --hard-dereference --file=$ftmp";
        printlog "Lanzando: $ccmd\n";
        $res = `$ccmd 2>&1 1>/dev/null`; # esta vez capturamos el STDERR
                                         #porque da mas informacion.
        printlog "Resultado de la importación:\n$res\n";

        # Por ultimo, colocamos el fichero de entrada en su sitio
        $ccmd = "mv $ftmp $param{'input'}";
        printlog "Lanzando: $ccmd\n";
        $res = `$ccmd`;
    } else {
        # No se ha podido localizar el fichero a importar
        $error = 8;
    }
}



#----------- Programa principal (main) -----------
# Aqui empieza la ejecución del script

%param = ();
```

```
$error = 0;
$flaglog = 0;  #false

recogeParam;

if ($error == 0) {
	# Los parámetros son sintácticamente correctos.
	if ($flaglog) {
		# Se ha elegido la creación de un log.
		if (open LOG, ">$param{'log'}") {
			cabeceraLog;
		} else {
			$error = 2;
		}
	}
}

if ($error == 0) {
	if ($param{'accion'} eq "ldir") {
		creaListaDir;
	} elsif ($param{'accion'} eq "exp") {
		exportar;
	} elsif ($param{'accion'} eq "imp") {
		importar;
	}
}

if ($error > 0) {
	# Se ha producido algún tipo de error

	if ($error == 1) {
		# Error de sintaxis.
		print "ERROR!! de sintaxis en la llamada al script\n";
		print "Debería pedir ayuda con \"psync.pl --help\"\n\n";
		printlog "\nERROR!! de sintaxis en la llamada al script\n";
		printlog "Debería pedir ayuda con \"psync.pl --help\"\n\n";
	}

	if ($error == 2) {
		# Error al crear el fichero de Log.
		printlog "ERROR!! No se ha podido crear el fichero de log
($param{'log'})\n\n";
		print "\nERROR!! No se ha podido crear el fichero de log
($param{'log'})\n\n";
	}

	if ($error == 3) {
		# Error al crear el fichero de salida.
		printlog "ERROR!! No se ha podido crear el fichero de salida
($param{'output'})\n\n";
		print "\nERROR!! No se ha podido crear el fichero de salida
($param{'output'})\n\n";
	}

	if ($error == 4) {
		# Error al acceder al fichero de directorios favoritos.
		printlog "ERROR!! No se ha podido acceder al fichero de favoritos
$param{'favorites'}.\n\n";
		print "\nERROR!! No se ha podido acceder al fichero de favoritos
$param{'favorites'}.\n\n";
	}

	if ($error == 5) {
```

```
            # Error al acceder al fichero de directorios direcorios.
            printlog "ERROR!! No se ha podido acceder al fichero de directorios
$param{'input'}.\n\n";
            print "\nERROR!! No se ha podido acceder al fichero de directorios
$param{'input'}.\n\n";
      }

      if ($error == 6) {
            #Error: hay alguna ruta a exportar que no existe
            printlog "ERROR!! Alguno de los directorios a exportar no existe.\n\n";
            print "\nERROR!! Alguno de los directorios a exportar no existe.\n\n";
      }

      if ($error == 7) {
            #Error al generar el fichero de exportacion
            printlog "ERROR!! No se ha podido generar el fichero
$param{'output'}.\n\n";
            print "\nERROR!! No se ha podido generar el fichero
$param{'output'}.\n\n";
      }

      if ($error == 8) {
            #Error al acceder al fichero de importacion
            printlog "ERROR!! No se ha podido acceder al fichero de datos a importar
$param{'input'}.\n\n";
            print "\nERROR!! No se ha podido acceder al fichero de datos a importar
$param{'input'}.\n\n";
      }
      if ($error == 9) {
            #Error al acceder al fichero de History
            printlog "ERROR!! No se ha podido acceder al fichero de History.\n\n";
            print "\nERROR!! No se ha podido acceder al fichero de History.\n\n";
      }

      if ($error == 10) {
            # Se ha pedido la ayuda.
            muestraAyuda;
      }

      if ($error == 11) {
            # Error al acceder al fichero ejecutable.
            printlog "ERROR!! No se ha podido acceder al fichero ejecutable
$param{'lib-dependences'}.\n\n";
            print "\nERROR!! No se ha podido acceder al fichero ejecutable
$param{'lib-dependences'}.\n\n";
      }

      if ($error == 12) {
            # Error al acceder al fichero makefile.
            printlog "ERROR!! No se ha podido acceder al fichero makefile
$param{'make-calls'}.\n\n";
            print "\nERROR!! No se ha podido acceder al fichero makefile $param{'make-
calls'}.\n\n";
      }
}

if ($flaglog) {
      pieLog;
      close LOG;
}
```