# PRÁCTICAS DE ADMINISTRACIÓN AVANZADA


(francisco.arguello@usc.es)

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela
15782 Santiago (España)

22 de enero de 2020

# Índice

# 1. Virtualización

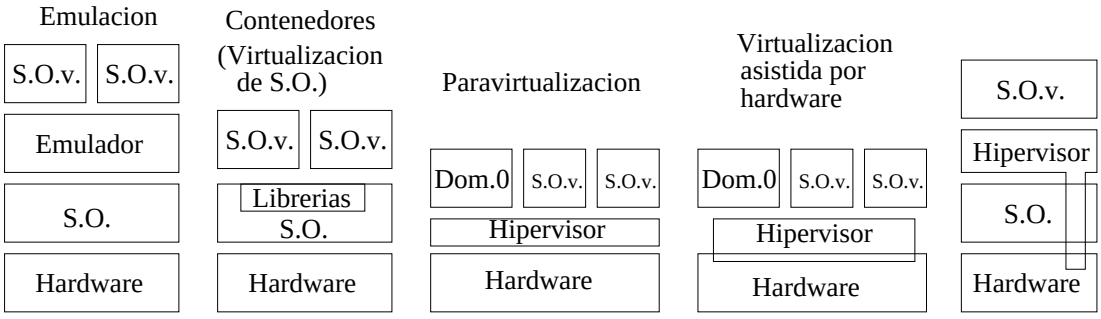
## 1.1. Tipos de virtualización

La virtualización es la recreación a través de software de algún recurso hardware o software, como puede ser un ordenador, un sistema operativo, un dispositivo de almacenamiento o una red. Para comprender los problemas que tiene la ejecución de un sistema operativo virtualizado hay que tener en cuenta que el software puede trabajar en dos modos:

- Un modo núcleo o kernel que puede ejecutar todas las instrucciones de CPU, incluyendo instrucciones "privilegiadas" que tienen que ver con el manejo del hardware como gestión de memoria, interrupciones, entrada/salida, etc.
- Un modo usuario que ejecuta únicamente instrucciones necesarias para procesar datos. Las aplicaciones se ejecutan en este modo y sólo pueden hacer uso del hardware solicitando al kernel una llamada al sistema.

Un procesador sin extensiones específicas carece de capacidad para ejecutar varios sistemas operativos a la vez, puesto que el modo núcleo afecta al estado del procesador y ello daría lugar a inconsistencias. En consecuencia, solo las instrucciones de modo usuario del sistema operativo virtualizado pueden ejecutarse sin problemas sobre el hardware. En cambio, las instrucciones del modo núcleo han de ser capturadas y gestionadas de alguna forma.

Así, existen cuatro principales tipos de virtualización:




### 1.1.1. Emulación

Un emulador funciona típicamente como un intérprete que va leyendo las instrucciones del sistema operativo virtual (*guest*) y traduciéndolas a instrucciones que pueden ejecutarse sin problemas sobre el sistema operativo anfitrión (*host*). De esta forma el emulador proporciona al sistema operativo virtual un entorno de ejecución simulado. El emulador en sí mismo se ejecuta como cualquier otra aplicación sobre el sistema operativo anfitrión. El hardware emulado puede coincidir o no con la máquina real sobre la que se ejecuta el emulador.

Las primeras versiones de los programas de virtualización típicos eran de este tipo, p.e., Qemu, Virtualbox, VirtualPC, VMware, así como infinidad de simuladores para juegos (mame, gameboy, nes, etc). Uno de los primeros sistemas de virtualización x86 fue desarrollado por VMware empleando traducción binaria. Esta aproximación examina el código ejecutable del sistema operativo virtual para encontrar instrucciones "inseguras", traducirlas por sus equivalentes "seguras" y luego ejecutar el código traducido.

### 1.1.2. Contenedores (Virtualización de Sistema Operativo)

En este tipo de virtualización se ejecuta únicamente un kernel, el del anfitrión, que también atiende a las llamadas de los sistemas virtuales. El software de virtualización (usualmente denominado de contenedor) utiliza un conjunto de librerías que traducen las llamadas del sistema operativo virtual al formato adecuado para ser ejecutadas sobre el hardware utilizando el sistema operativo anfitrión. El entorno proporcionado para el contenedor puede coincidir o no con la máquina real sobre la que se ejecuta. En el caso de que no coincida, el traductor tendrá bastante complejidad. Ejemplos de programas que realizan este tipo de virtualización son LXC, OpenVZ y Docker (Wine hace algo similar en el espacio de usuario).

### 1.1.3. Paravirtualización

En este caso se modifica el código fuente del sistema operativo virtual para sustituir todas las operaciones problemáticas. El objetivo de las modificaciones es que el kernel del sistema operativo virtual coopere con el kernel del sistema operativo anfitrión. De esta forma se consigue que el hardware que ven las maquinas virtuales sea muy similar al hardware real. El software de virtualización (usualmente denominado hipervisor) es una fina capa de software que corre directamente sobre el hardware, multiplexa los accesos de los distintos sistemas operativos al hardware, además de proporcionar un entorno para su control y gestión (el dominio 0). Proporciona un alto rendimiento pero tiene la desventaja de requerir modificaciones al kernel de los sistemas operativos virtuales. Una aplicación que realiza paravirtualización es Xen.

### 1.1.4. Virtualización asistida por hardware

También se denomina virtualización nativa[1]. En este tipo de virtualización las instrucciones del sistema operativo virtual se ejecutan sin modificación alguna directamente sobre el hardware. El procesador dispone de las extensiones hardware

[1]La virtualización asistida por hardware se denomina a veces virtualización completa, aunque este último término también puede referirse a la emulación.

necesarias para poder ejecutar varios sistemas operativos a la vez. De esta forma, el efecto de las operaciones privilegiadas se mantiene dentro de la máquina virtual, sin alterar el estado de las otras máquinas virtuales o del hardware. Un hipervisor multiplexa y controla la utilización del hardware.

Este tipo de virtualización ha sido incluida desde hace más de 50 años en los mainframes de IBM. Sobre los procesadores x86, la virtualización asistida por hardware no se ha podido realizar hasta 2005-2006 con la incorporación de las extensiones AMD-V e Intel VT. Usualmente en esta arquitectura sólo el procesador se virtualiza por hardware, mientras que el resto de los elementos, como las tarjetas de expansión (gráfica, ethernet, etc), siguen siendo emuladas, en contraste con la arquitectura de IBM en la que todos los elementos hardware son virtualizados.

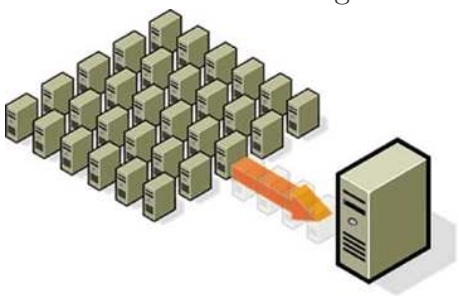
Las aplicaciones Xen y KVM (incluido en el kernel desde la versión 2.6.20) pueden usar estas extensiones para realizar virtualización asistida por harware sobre los procesadores x86. También las pueden utilizar aplicaciones que ejecutan máquinas virtuales por encima del sistema operativo, como Virtualbox, VirtualPC, entre otras.[2] [3] En este caso, el hipervisor comprende un módulo incluido en el kernel.

Podemos determinar que nuestro procesador tiene extensiones de virtualización (Intel VT o AMD-V) si recibimos respuesta para los siguientes comandos (muchas veces hay que activarlas en la BIOS):

```
$ cat /proc/cpuinfo | grep svm   (para procesadores AMD)
$ cat /proc/cpuinfo | grep vmx   (para procesadores Intel)
```

## 1.2. Ventajas de la virtualización

Las ventajas de la consolidación de servidores (reducción de servidores físicos e integración) mediante virtualización son las siguientes:[4]



- **Ahorro de costes**. Podemos tener un reducido número de servidores, aunque más potentes, en vez de infinidad de servidores individuales. Con esto

[2] Se denominan hipervisores de tipo 2, en contraste con los hipervisores de tipo 1.

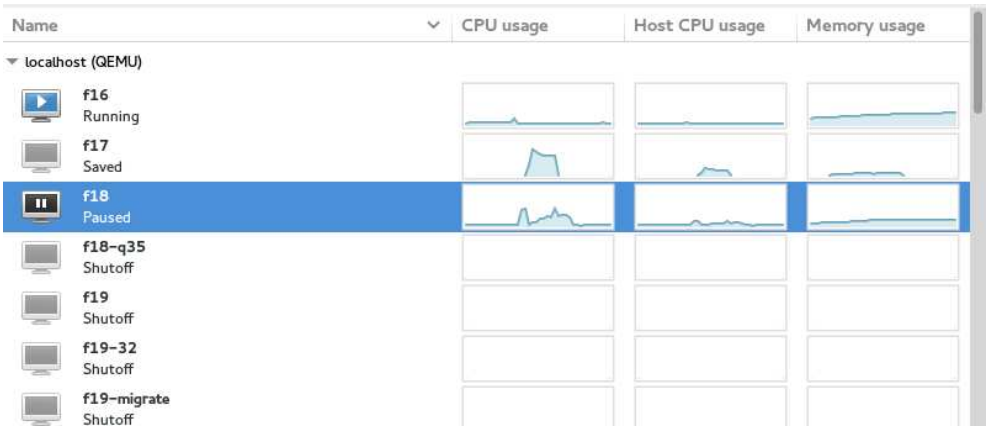
[3] En los procesadores x86, bajo determinadas condiciones la virtualización software puede ser más rápida que la hardware, como lo demuestra VMware.

[4] http://www.niux.com.ar/usos-virtualizacion-y-para-que-sirve-la-virtualizacion/.

conseguimos ahorro de espacio, mantenimiento y consumo de energía. Los recursos fisicos del conjunto de servidores se pueden repartir dinámicamente, por ejemplo, se puede otorgar más CPU, memoria, almacenamiento o ancho de banda a la máquina virtual que lo necesite.

- **Administración simplificada**. Desde la consola del servidor podemos gestionar cada máquina virtual: pararla, reiniciarla, actualizarla, borrarla en caso de problemas, hacer backups, aumentar o reducir los recursos para una determinada máquina, etc. Además, se puede mantener una copia en producción mientras en otra copia se realiza el mantenimiento. En caso de ofrecer un servicio de *hosting*, cada cliente tendrá asignada una o más máquinas virtuales, en las que podrá operar con permisos de administración.
- **Seguridad**. La virtualización proporciona aislamiento entre la máquina huésped y las máquinas virtuales, así como entre una máquina virtual y las demás. Cualquier problema de seguridad quedará restringido a una máquina virtual concreta.
- **Aprovechamiento de aplicaciones antiguas**. Una de las ventajas de la virtualización es la posibilidad de conservar aplicaciones que funcionan en sistemas antiguos y aun así modernizar la infraestructura informática de la empresa. Esa aplicación puede "sobrevivir" en una máquina virtual independiente sin que haga falta conservar el ordenador antiguo.

Ejemplo de consola de gestión de virt-manager:




Ejemplo de consola de gestión de Xen:

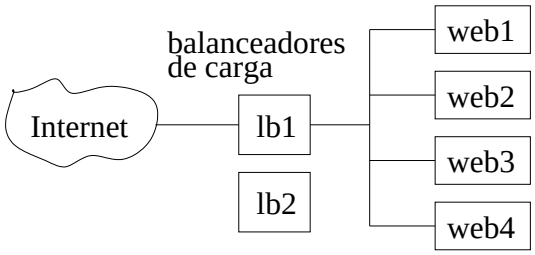
```
Name                 Id  Mem(MB)  CPU  State  Time(s)  Console
Domain-0              0      123    0  r----     11.6
ttylinux1             1        7    0  -b---      1.7    9601
ttylinux2             2        7    0  -b---      1.7    9602
```

Ejemplo de consola de gestión de OpenVZ:

```
CTID      NPROC STATUS  IP_ADDR         HOSTNAME
 555          7 running 192.168.0.10    pepito.com
```

## 1.3. Linux Virtual Server (LVS)

LVS[5] es un entorno de balanceo de carga en sistemas Linux orientado a proporcionar altas prestaciones en entornos con tecnología cluster. El esquema típico de funcionamiento es el siguiente: (1) un cliente solicita un recurso, (2) el balanceador de carga reenvía la solicitud a uno de los servidores, y (3) el servidor responde al cliente.




Hay tres principales alternativas:

1. **LVS-DR (Direct Routing)**. El balanceador de carga reenvía los paquetes a los servidores mediante la modificación de la dirección MAC de las tramas de datos con las del servidor seleccionado.

2. **LVS-NAT (Network Address Translation)**. Los paquetes son redirigidos usando NAT (traducción de direcciones). Esto es, se modifican las direcciones IP de los datagramas con las del servidor seleccionado.

3. **LVS-TUN (Tunneling)**. Se generan túneles desde el balanceador de la carga a los servidores. Para ello los datagramas se encansulan con nuevas cabeceras que incluyen las direcciones de los servidores destino.

Una solución sencilla para LVS-NAT se basa en el uso del comando *iptables* con el paquete de estadística, que alterna las direcciones IP cada cierto número de paquetes. En el siguiente ejemplo se cambian las direcciones y puertos de los datagramas destinados al servidor web: de cada dos datagramas, el primero se cambia a 10.0.0.5 y el segundo a 10.0.0.6 (el puerto también se cambia de 8000 a 80).

```
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8000 -m state --state NEW \
   -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 10.0.0.5:80
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8000 -m state --state NEW \
   -m statistic --mode nth --every 2 --packet 1 -j DNAT --to-destination 10.0.0.6:80
```

[5] http://www.austintek.com/LVS/LVS-HOWTO/.

# 2. Emulación (Qemu/KVM y Virtualbox)

## 2.1. Qemu/KVM

Qemu (o KVM) es un emulador para la arquitectura PC, incluyendo el procesador y varios periféricos. En concreto, Qemu simula los siguientes periféricos:

| |
|---|
| Puente host-PCI (i440FX) y puente PCI-ISA (PIIX3) |
| Tarjeta gráfica PCI VGA (Cirrus CLGD 5446) o VESA |
| Ratón y teclado PS/2 |
| 2 interfaces PCI IDE para discos duros y CDROM |
| Tarjetas de red PCI (NE2000 y otras) |
| Puertos serie, paralelo y USB |
| Tarjeta de sonido (Soundblaster 16 y otras) |

La invocación del comando (dependiendo de la distribución) es:

```
$ kvm [options] [disk_image]
$ qemu-system-i386 [options] [disk_image]
$ qemu-system-x86_64 [options] [disk_image]
```

### 2.1.1. CPU, Memoria y Discos

`-m` *megs* (tamaño de la memoria RAM)
`-hda` *file*, `-hdb` *file*, `-hdc` *file*, `-hdd` *file* (discos duros)
`-cdrom` *file* (CDROM)
`-no-kvm` (desactiva la virtualización asistida por hardware)
`-snapshot` (la imagen original no se modifica, sino que las modificaciones se escriben en ficheros temporales, en vez de en los discos. La modificaciones se pueden escribir a la imagen original con CNTL-a s)

### 2.1.2. Teclas de escape

`Ctrl-Alt` (captura/libera el ratón y la consola)
`Ctrl-Alt-f` (conmuta a pantalla completa)
`Ctrl-Alt-`*n* (conmuta a la consola *n*, 1=normal, 2=monitor, 3=puerto serie)
`Ctrl-a h` (ayuda)
`Ctrl-a x` (sale del emulador)
`Ctrl-a s` (graba las modificaciones en la imagen cuando se usan snapshots)

### 2.1.3. Modo monitor

`info` *subcommand* (da información del componente especificado, por ejemplo, `network`, `block`, `registers`, `history`, `pci`, `usb`, `mice`, `snapshots`, etc)
`q` o `quit` (salir)

`eject [-f]` *device*
`change` *diskdevice filename* (cambia los parámetros de un dispositivo, por ejemplo, el fichero imagen). Para cambiar de CD: `change cdrom` *image.iso*
`savevm` *[tag|id]* (crea un snapshot, esto es, graba el estado de la máquina virtual completa, para después poder reiniciarla en ese punto de la simulación).
`loadvm` *tag|id* (carga un snapshot)
`screendump` *filename* (graba la imagen de la ventana en un fichero ppm)

### 2.1.4. Grabación del estado de la máquina virtual

KVM permite generar snapshots de las máquinas virtuales para continuar posteriormente la ejecución en el punto en las que las hemos parado. La condición es que el formato de fichero de la máquina virtual los admita (por ejemplo, *qcow2*). Para realizar la conversión disponemos del siguiente comando:

```
$ qemu-img convert -O qcow2 debian1.img debian1.qcow2
```

Podemos disponer de múltiples snapshots con su correspondiente ID. El procedimiento es el siguiente:

1. Cuando estamos ejecutando la máquina virtual en KVM y queramos crear un snapshop, pasamos al modo monitor tecleando: CNTL-ALT 2.

   A continuación, paramos la emulación y creamos el snapshot:

   ```
   (qemu) stop
   (qemu) savevm ID
   (qemu) quit
   ```

2. Cuando queramos recuperar el snapshot, pasamos al modo monitor con CNTL-ALT 2 y tecleamos:

   ```
   (qemu) loadvm ID
   (qemu) cont
   ```

   Volvemos al modo de emulación tecleando CNTL-ALT 1.

### 2.1.5. Migración entre dos hosts

Esta operación es para transferir un máquina virtual entre dos hosts, sin parar en ningún momento la ejecución de la máquina virtual. Los pasos son los siguientes:

1. Iniciamos el Qemu destino con los mismos parámetros que el Qemu origen y con el puerto que deseamos para la migración. Un condición importante es que el Qemu destino debe tener acceso al disco en el cual está almacenada la imagen que se va a migrar.

   ```
   $ kvm -incoming tcp:0:4444
   ```

2. El Qemu origen lo iniciamos normalmente:

   `$ kvm maquina_virtual.img`

3. Para la migración, en el Qemu origen pasamos al modo monitor (CNTL-ALT 2) y tecleamos lo siguiente:

   `(qemu) migrate -d tcp:127.0.0.1:4444 (qemu) info migrate`

   Si todo ha ido bien, la emulación deberá continuar en el Qemu destino.

## 2.2. Imagen empleada en la prácticas

Usaremos Qemu/KVM o Virtualbox en las prácticas de esta asignatura.

### 2.2.1. KVM/Qemu

Nos descargamos de la USV virtual el fichero `debian-sarge.img.gz` y lo descomprimimos, denominádo a la imagen *debian1.img*. Ejecutamos el comando KVM de la siguiente forma:

```
$ kvm -name "debian1" debian1.img -vga cirrus -m 512 -device ne2k_pci,netdev=net0 -netdev \
  user,id=net0,hostfwd=tcp::2200-:22,hostfwd=tcp::2300-:2300,hostfwd=tcp::8000-:80
```

Hemos iniciado la máquina virtual con una red de usuario (la conexión a la red será proporcionada por el propio emulador) y hemos redireccionado los puertos ssh (22), telnet (23) y http (80), para poder conectarnos desde el exterior a la máquina virtual.

Por defecto, las direcciones proporcionadas por el servidor DHCP integrado de KVM estarán en la red 10.0.2.0/24, siendo 10.0.2.15 la primera máquina virtual y 10.0.2.2 la pasarela.

### 2.2.2. Virtualbox

Alternativamente, las prácticas pueden realizarse con Virtualbox. Nos descargamos de la USV virtual el fichero `debian-sarge.vdi.gz` y lo descomprimimos. Cuando construyamos la imagen con la GUI de Virtualbox, necesitamos hacer los siguientes ajustes:

1. **Máquina debian1**. Generamos una máquina debian de 32 bits denominada *debian1* a partir del archivo `debian-sarge.vdi` y 512 MB de RAM.

2. **Disco**. La imagen debian proporcionada trabaja con discos IDE (y no SATA). Por tanto en virtualbox en el menú de almacenamiento borramos el disco SATA y le añadimos un disco IDE.

3. **Red**. El interface conectado al exterior será de tipo *NAT*.

   Por defecto, al igual que en KVM, las direcciones proporcionadas por el servidor DHCP de Virtualbox estarán en la red 10.0.2.0/24, siendo 10.0.2.15 la máquina virtual y 10.0.2.2 la pasarela. Adicionalmente, nos interesa hacer redirecciones de los puertos ssh (22), telnet (23) y ssh (80):

   ```
   $ VBoxManage modifyvm debian1 --natpf1 "guestssh,tcp,127.0.0.1,2200,,22"
   $ VBoxManage modifyvm debian1 --natpf1 "guesttelnet,tcp,127.0.0.1,2300,,2300"
   $ VBoxManage modifyvm debian1 --natpf1 "guesthttp,tcp,127.0.0.1,8000,,80"
   ```

4. **Ratón**. Una vez iniciada la máquina, si el ratón no se mueve o lo hace erráticamente, podemos eliminar la integración del ratón en virtualbox, pinchando con el botón derecho del ratón en los iconos de la parte inferior de la ventana.

### 2.2.3. Máquina virtual Debian sarge

Una vez arrancada la máquina virtual, tenemos los usuarios *root* y *debian* ambos con la clave *debian*. Si como root no nos deja usar la pantalla gráfica, tecleamos como el primer usuario que entró en la máquina **xhost +**. Además, disponemos del editor de textos gráfico *nedit*.

La red no está configurada, pero vamos al directorio de configuración:

```
$ cd /etc/network
```

Ahora editamos `interfaces` y añadimos:

```
auto eth0
iface eth0 inet dhcp
```

Reinciamos la red con:

```
$ /etc/init.d/networking restart
```

Y comprobamos que tenemos acceso a Internet con el siguiente comando, que nos debería devolver la IP de este servidor web:

```
$ host www.usc.es
```

Es interesante poder conectarnos de forma de remota desde un PC para hacer operaciones de edición y de copiar/pegar y también para la transferencia de ficheros. Esto lo haremos a partir del puerto 2200 que hemos habilitado:

```
$ ssh -p 2200 -X debian@localhost
$ sftp -P 2200 debian@localhost
```

## 2.3. Resolución de problemas

- **Proxy**. Para conectarse al exterior de la USC desde las máquinas virtuales de los PCs de cable del laboratorio es necesario activar el proxy de la USC, bien con las opciones del navegador o con los comandos de terminal. En un terminal de debian-sarge, teclearíamos:

  ```
  $ export http_proxy=http://proxy2.usc.es:8080
  $ export https_proxy=http://proxy2.usc.es:8080
  $ export ftp_proxy=http://proxy2.usc.es:8080
  ```

  La configuración solo tiene lugar mientras estemos en la sesión de bash, por lo que si queremos hacerlo permanente, introduciremos estas líneas en el fichero `.bashrc` del usuario o de root.

  El navegador *firefox* tiene su propio menú de configuración de red en el que se puede poner el proxy.

  No se necesita en los portátiles ni tampoco si trabajamos desde casa.

- **No arranca kvm**. En el caso de que KVM no arranque, debido a que estamos dentro de una simulación virtualbox o no tenemos virtualización hardware, iniciar con:

  ```
  $ kvm -no-kvm ...      $ qemu-system-i386 ...
  ```

- **Pantalla**. Si la máquina virtual inicia, pero sin pantalla gráfica, se puede editar en la máquina virtual el fichero */etc/X11/XF86Config-4* y sustituir el *Driver cirrus* por el *Driver vesa*.

- **Ratón**. La velocidad del ratón se puede ajustar en el menú del entorno gráfico de la máquina virtual. Para capturar el ratón en Linux Mint hay que teclear CNT-ALT-G. Si el ratón no se mueve o lo hace erráticamente, podemos eliminar la integración del ratón en virtualbox, pinchando con el botón derecho del ratón en los iconos de la parte inferior de la ventana.

- **Interface de red**. En caso de que la máquina virtual no reconozca los interfaces de red, podemos probar indicándole a kvm otro modelo. La lista se obtiene con `kvm -device ?`. Por ejemplo:

  ```
  $ kvm imagen.iso -device ne2k_pci,netdev=net0 ...
  $ kvm imagen.iso -device pcnet,netdev=net0 ...
  ```

  Puede ser necesario cargar un módulo en la máquina virtual con:

  ```
  $ modprobe ne2k-pci      $ modprobe pcnet32.
  ```

- **Formato raw**. Para evitar el mensaje *WARNING: Specify the 'raw' format explicitly to remove the restrictions*, podemos indicar el fichero de máquina virtual con la opción `-drive file=debian1.img,format=raw`.

# 3. Creación de imágenes para virtualización

## 3.1. Comandos básicos

A diferencia del sistema operativo real que se instala directamente sobre el ordenador, los sistemas operativos virtuales se instalan sobre imágenes, que pueden estar contenidas en **ficheros, directorios** o **volúmenes lógicos LVM**. Por ello, la gestión de los sistemas operativos virtuales es muy sencilla (una vez que se tiene un máster, basta con sacar las copias que se necesiten). Veamos algunos comandos básicos para trabajar con estas imágenes.

1. El comando `dd` permite realizar copias, conversiones y formateos a bajo nivel. El formato básico y algunos ejemplos son:

   `dd if=`*origen* `of=`*destino* `[`*opciones*`]`
   `dd if=`*/dev/cdrom* `of=`*fichero* (copia un CD a fichero)
   `dd if=`*/dev/hda* `of=`*/dev/hdb* `bs=`*1k* (copia discos duros bloque a bloque)
   `dd if=`*/dev/zero* `of=`*imagen.iso* `bs=`*1k* `count=`*1* `seek=`*60k* (crea una imagen de disco duro de tamaño inicial un bloque de 1k y con espacio vacío para 60k bloques (60 MB), cuyo tamaño va aumentando conforme se necesite).

   Más en detalle, el comando *dd* tiene las opciones:

   | | |
   |---|---|
   | bs=BYTES | lee y escribe grupos de BYTES (bloques) de cada vez |
   | count=BLOQUES | copia solo el número de BLOQUES especificado |
   | seek=BLOQUES | BLOQUES vacíos al compienzo de fichero. |

   El tamaño de un bloque usualmente es 4k en los sistemas de ficheros actuales (ext4 y NTFS) y de 1k en los sistemas antiguos (ext3 e ISO9660 (CD y DVD)), por lo que conviene usar estos tamaños para ajustarnos a la capacidad exacta de los dispositivos.[6]

   Una vez creada la imagen de un disco, esta puede formatearse inmediatamente o bien particionarse, para luego formatear las particiones.

2. El comando `fdisk` nos permite crear particiones en un disco. El formato es `fdisk` *dispositivo* y una vez entrado en un submenú las opciones más comunes son:

   `m` (ayuda)
   `p` (ver tabla de particiones)
   `n` (nueva partición, en la cabecera se indica la unidad de tamaño en bytes)
   `t` (cambiar la etiqueta que especifica el formato de la partición)
   `w` (escribir los cambios y salir)

[6] El tamaño de bloque puede obtenerse con: `dumpe2fs /dev/XXXX | grep 'Block size'`, donde XXXX es una partición de disco.

Por ejemplo, si particionamos el disco asociado al dispositivo */dev/hda*, las particiones serán */dev/hda1*, */dev/hda2*, etc.

En los PCs modernos los discos duros se suelen denominar *sda, sdb, ...* (SATA disk) frente a los discos duros antiguos *hda, hdb, ...* (hard disk, de tipo IDE) y los CD/DVD *scd0, scd1, ....*

Estas particiones puede formatearse, por ejemplo, `mkfs.ext3` */dev/hda1* (para los sistemas de ficheros) o `mkswap` */dev/hda2* y `swapon` */dev/hda2* (para el área de intercambio).

3. El comando `parted` nos permite cambiar el tamaño de una partición. Hay que tener en cuenta que solo puede ampliar una partición si hay espacio libre adyacente, pues no es tan flexible como LVM (gestor de volúmenes lógicos). El formato es `parted` *dispositivo* y una vez entrado en un submenú las opciones más comunes son:

   `print` (muestra las particiones)
   `rm` (borra una partición)
   `resize` (cambia de tamaño una partición, se puede dar en M)
   `mkfs` (formatea una partición)
   `check` (testea una partición)
   `quit` (sale de la aplicación)

### 3.1.1. Instalación de un Linux sencillo

Vamos a comenzar creando una imagen para ttylinux, una de las distribuciones Linux más pequeñas. Desde la página de la materia o bien desde la página del proyecto[7] nos traemos el archivo *bootcd-i486-8.1.iso.gz* y lo descomprimimos. El fichero es una imagen de CDROM, así que vamos a crear y a formatear una imagen de disco duro RAW de 10 MB para instalarlo. Los comandos son los siguientes:

```
PC REAL$ bzip2 -d bootcd-i486-8.1.iso.bz2
PC REAL$ dd if=/dev/zero of=ttylinux.iso bs=1k seek=10k count=1
```

Ahora creamos una partición para contener todo el sistema operativo:

```
PC REAL$ fdisk ttylinux.iso
fdisk: m
fdisk: p
fdisk: n p 1 1
fdisk: p
fdisk: m
fdisk: w
```

[7] http://ttylinux.net.

NOTA: En Virtualbox convertimos la imagen RAW al formato VDI con:

```
PC REAL$ VBoxManage convertfromraw ttylinux.iso ttylinux.vdi
```

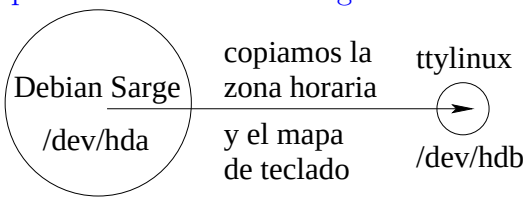
y la incluimos en la configuración de almacenamiento como disco IDE primario master y al CDROM como IDE secundario master. No olvidarse de retirar el CDROM una vez que el sistema operativo esté instalado en el disco duro.

Arrancamos Qemu e instalamos ttylinux en el disco duro. Para ello usaremos un instalador proporcionado por la distribución que se limita a copiar los ficheros desde el CD (dispositivo */dev/hdc*) al disco duro (dispositivo */dev/hda1*) y a crear un menú de arranque:[8] Fijarse que el instalador nos dice que el sistema de ficheros se instalará en */dev/hda1* y el cargador (del menú de arranque) en */dev/hda*.

```
PC REAL$ kvm ttylinux.iso -cdrom bootcd-i486-8.1.iso -boot d
TTYLinux$ ttylinux-installer -m /dev/hdc /dev/hda1
TTYLinux$ poweroff
```

NOTA: En Virtualbox, dependiendo de donde hayamos colocado el CDROM la unidad podría ser */dev/hdb* en vez de */dev/hdc*.

La instalación básica ya estaría completa, pero vamos a configurar la imagen estableciendo la zona horaria y el teclado español. Lo hacemos copiando los ficheros de una distribución Debian sarge. Para poder acceder desde Debian (dispositivo */dev/hda*) a la otra imagen (dispositivo */dev/hdb*) tenemos que montar esta última en la máquina virtual Debian sarge.




```
PC REAL$ kvm -name "debian1" debian1.img -hdb ttylinux.iso \
  -vga cirrus -m 512 -device ne2k_pci,netdev=net0 -netdev \
  user,id=net0,hostfwd=tcp::2200-:22,hostfwd=tcp::2300-:2300,hostfwd=tcp::8000-:80 &
Debian$ mkdir /mnt/ttylinux
Debian$ mount /dev/hdb1 /mnt/ttylinux
Debian$ cp /etc/localtime /mnt/ttylinux/etc/localtime
Debian$ /mnt/ttylinux/bin/dumpkmap > /mnt/ttylinux/etc/i18n/kmap
Debian$ umount /mnt/ttylinux
Debian$ poweroff
```

Ya podemos utilizar la imagen de ttylinux en el emulador:

```
PC REAL$ kvm ttylinux.iso
```

[8] El mapa de teclado que viene por defecto es inglés, por lo que debemos tener en cuenta que el caracter - está en la tecla ´ (debajo de ?) y el caracter / en la tecla -.

Para terminar, vamos a configurar la red. La podríamos configurar con los comandos usuales (`ifconfig`, etc), pero la haremos permanente editando el fichero `/etc/sysconfig/network-scripts/ifcfg-eth0`, que es la configuración típica de la distribución RedHat. En este archivo podemos indicar que se use el servidor DHCP integrado de KVM:

```
ENABLE=yes
DHCP=yes
```

Alternativamente podemos configurar estáticamente la IP y gateway que proporcionan KVM y virtualbox:

```
ENABLE=yes
DHCP=no
IP=10.0.2.15
gateway=10.0.2.2
...
```

Para el servicio de nombres creamos el fichero `/etc/resolv.conf`:

```
nameserver 193.144.75.9  # servidor de la USC
nameserver 8.8.8.8       # servidor de Google
nameserver 10.0.2.2      # servidor integrado de KVM/Virtualbox
```

y reiniciamos el servicio de red con:

```
TTYLinux$ service network restart.
```

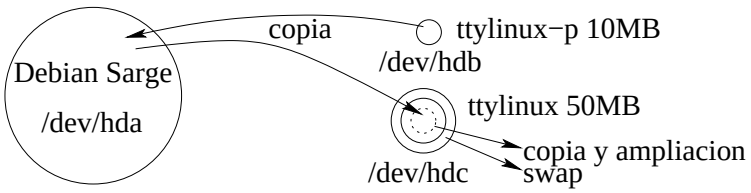
Podemos comprobar que la red funciona con:

```
TTYLinux$ ping 10.0.2.2
TTYLinux$ nslookup www.usc.es
```

⇒ **ENTREGA 1:** ttylinux con teclado español y la red configurada.

### 3.1.2. Cambio del tamaño de una imagen

En este apartado vamos a copiar y a ampliar el tamaño de una imagen ttylinux utilizando como auxiliar el S.O. Debian sarge. Para ello, crearemos una imagen RAW vacía del tamaño considerado, con Debian sarge copiaremos la imagen de partida y reajustaremos el tamaño de la partición 1. Por último, en el espacio de disco no utilizado aprovecharemos para crear una partición nueva de *swap*.



Primero creamos la imagen vacía de 50 MB y descargamos de la USC virtual el archivo *ttylinux-p.iso.bz2* [9]. A continuación arrancamos un PC virtual Debian sarge en el primer disco duro, cargamos la imagen ttylinux origen como segundo disco duro (*-hdb*) y como tercer disco duro (*-hdc*) el destino vacío:

```
PC REAL$ dd if=/dev/zero of=ttylinux-50.iso bs=1k seek=50k count=1
PC REAL$ kvm -name "debian1" debian1.img -hdb ttylinux-p.iso \
  -hdc ttylinux-50.iso -vga cirrus -m 512 -device ne2k_pci,netdev=net0 -netdev \
   user,id=net0,hostfwd=tcp::2200-:22,hostfwd=tcp::2300-:2300,hostfwd=tcp::8000-:80 &
```

NOTA: En Virtualbox debemos convertir ambos ficheros a formato VDI antes de cargarlos como unidades de discos IDE adicionales:

```
PC REAL$ VBoxManage convertfromraw ttylinux-p.iso ttylinux-p.vdi
PC REAL$ VBoxManage convertfromraw ttylinux-50.iso ttylinux-50.vdi
```

Copiamos la imagen origen en la imagen vacía en bloques de 1024 bytes:

```
Debian$ dd if=/dev/hdb of=/dev/hdc bs=1024
```

Con el programa `parted` ajustamos el tamaño de la partición 1 del destino:

```
Debian$ parted /dev/hdc
parted: print
parted: resize
parted:   Partition number? 1
parted:   Start?  [0,0308]?
parted:   End?    [7,8438]? 40M
parted: quit
```

Por último, aprovechamos el espacio vacío para crear una partición de *swap*:

```
Debian$ fdisk /dev/hdc
fdisk: m
fdisk: p
fdisk: n p 2 6
fdisk: t 2 L 82
fdisk: p
fdisk: w
Debian$ halt
```

[9] OJO, hay una incompatibilidad de versiones entre el comando `mkfs.ext2` de ttylinux y el comando `parted` de Debian, que se traduce en el *Error: Filesystem has incompatible feature enabled*. El error se soluciona formateando las particiones desde Debian y comentando las líneas 260-262 del script `ttylinux-installer` de ttylinux en donde se hace una llamada al comando `mke2fs`. Para evitar tener que hacer estas correcciones, se proporciona una imagen *ttylinux-p.iso* con la incompatibilidad corregida.

Ya podemos utilizar la nueva imagen ampliada:

```
PC REAL$ kvm ttylinux-50.iso
TTYLinux$ df -h
```

Nos quedaría editar */etc/fstab* para aprovechar la partición de *swap*:

```
TTYLinux$ vi /etc/fstab
        /dev/hda2 none swap sw 0 0
```

y formatearla y activarla con: `mkswap /dev/hda2` y `swapon /dev/hda2`.

⇒ **ENTREGA 2:** ttylinux con tamaño ampliado y swap activado.

## 3.2. Creación de imágenes usando una jaula

Vamos a crear unas imágenes adecuadas para virtualización usando los comandos `debootstrap` y `chroot`. El comando `debootstrap` nos permite traer a nuestro ordenador el sistema de ficheros de alguna distribución Linux sin necesidad de instalación. La distribución se almacenará en un directorio y podremos ejecutarla sin peligro desde la distribución antigua con el comando `chroot`. Todas las operaciones que realizemos tendrán efectos internos al directorio, dentro de una *jaula*.

### 3.2.1. Obtención de la distribución Debian potato

(Si estamos en la red de cable de la USC para conectarnos al exterior desde la máquina virtual hay que activar el proxy):

```
Debian$ export ftp_proxy=http://proxy2.usc.es:8080
Debian$ export http_proxy=http://proxy2.usc.es:8080
Debian$ export https_proxy=http://proxy2.usc.es:8080
```

Desde Debian sarge (v3.2) probaremos la distribución Debian potato (v2.2):

```
Debian$ debootstrap --arch i386 potato /home/debian/potato \
   http://archive.debian.org/debian
```

Entramos en la jaula y a partir de aquí estaremos trabajando sobre *potato*.

```
Debian$ chroot /home/debian/potato
```

Antes que nada vamos a configurar la instalación de paquetes y a instalar algunos de ellos. Para ello realizamos las siguientes operaciones:

```
Potato$ dpkg-reconfigure -a
Potato$ vi /etc/apt/sources.list
          deb http://archive.debian.org/debian potato main
Potato$ apt-get update
Potato$ apt-get install telnetd ftpd apache
Potato$ adduser guest (con clave guest)
```

Normalmente con estas operaciones terminamos. Pero vamos a realizar algunas tareas que después nos serán útiles para la virtualización (si no las hacemos ahora, nos las pediría hacerlas después).

En concreto, vamos a editar dos ficheros, el primero el montado de dispositivos en el arranque (*/etc/fstab*) y el segundo la configuración de las direcciones de los interfaces de red (*/etc/network/interfaces*):

```
Potato$ vi /etc/fstab
          proc      /proc  proc defaults                  0 0
          /dev/hda1 /      ext3 defaults,errors=remount-ro 0 1
Potato$ vi /etc/network/interfaces
          # esta distribucion es antigua y no tiene: auto lo
          iface lo inet loopback
```

Por último, podemos ahorrar un poco de espacio de disco borrando de la cache los paquetes descargados:

```
Potato$ rm /var/cache/apt/archives/*.deb
```

Ya podemos salir de la jaula con `exit`.

### 3.2.2. Empaquetamiento de la imagen

Podemos crear un fichero comprimido con la imagen de la siguiente forma:

```
Debian$ cd /home/debian
Debian$ tar -jcvf potato.tar.bz2 potato
```

Y vamos a crear un fichero RAW que contenga la imagen. Como el tamaño del sistema de ficheros es de 47 MB vamos a crear una imagen de 100 MB. Luego la formateamos, la montamos, copiamos los ficheros y por último la desmontamos.

```
Debian$ dd if=/dev/zero of=potato.iso bs=1k seek=100k count=1
Debian$ mkfs.ext3 potato.iso
Debian$ mkdir /mnt/potato
Debian$ mount -o loop potato.iso /mnt/potato
Debian$ cp -R /home/debian/potato/* /mnt/potato
Debian$ umount /mnt/potato
```

Ya tenemos el sistema de ficheros de potato dentro de la imagen *potato.iso*. Esta imagen será usada en una práctica de Xen.

### 3.2.3. Instalación de un kernel

Pasamos ahora a instalar un kernel. Esto solo es necesario si queremos que la imagen sea utilizable en un emulador, pues el resto de los virtualizadores proporcionan su propio kernel. Para ello vamos al directorio en el cual está la distribución, entramos en la jaula *chroot* y nos traemos el paquete del kernel (respondiendo a las preguntas que nos hace que NO).

```
Debian$ chroot /home/debian/potato
Potato$ apt-get install kernel-image-2.2.19
Potato$ exit
Debian$ halt
```

Lo más complicado es instalar un gestor de arranque, pues no podemos hacerlo dentro de un fichero, sino que tenemos que cargar la imagen como disco controlado por un dispositivo. Para ello, primero salimos al PC real y creamos una imagen de disco duro vacío:

```
PC REAL$ dd if=/dev/zero of=potato_con_kernel.iso bs=1k seek=200k count=1
```

Ahora iniciamos Qemu con esta imagen como segundo disco duro:

```
PC REAL$ kvm -name "debian1" debian1.img -hdb potato_con_kernel.iso \
  -vga cirrus -m 512 -device ne2k_pci,netdev=net0 -netdev \
  user,id=net0,hostfwd=tcp::2200-:22,hostfwd=tcp::2300-:2300,hostfwd=tcp::8000-:80 &
```

NOTA: en Virtualbox debemos de convertir la imagen a formato VDI antes de cargarla como disco IDE:

```
PC REAL$ VBoxManage convertfromraw potato_con_kernel.iso potato_con_kernel.vdi
```

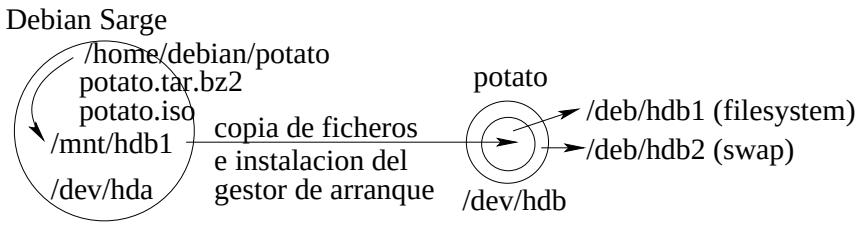
Ya dentro del Qemu creamos las particiones del disco duro con el comando `fdisk`. La primera será para el sistema de ficheros y la segunda para el *swap*.

NOTA: en Virtualbox el número de sectores puede variar, así que elegir un 90 % del tamaño del disco para la partición Linux y un 10 % para la de swap.

```
Debian$ fdisk /dev/hdb
fdisk: m
fdisk: p
fdisk: n p 1 (1-24)
fdisk: n p 2 (25-25)
fdisk: p
fdisk: m
fdisk: t 1 L 83
fdisk: t 2 L 82
fdisk: p
fdisk: m
fdisk: w
```

Formateamos la partición 1, la montamos y copiamos el sistema de ficheros:

```
Debian$ mkfs.ext2 /dev/hdb1
Debian$ mkdir /mnt/hdb1
Debian$ mount /dev/hdb1 /mnt/hdb1
Debian$ cp -R /home/debian/potato/* /mnt/hdb1
```




Añadimos una línea a */etc/fstab* para que se cargue el *swap*:

```
Debian$ vi /mnt/hdb1/etc/fstab
         /dev/hda2 none swap sw 0 0
```

Y configuramos el menú de arranque:

```
Debian$ mkdir /mnt/hdb1/boot/grub
Debian$ vi /mnt/hdb1/boot/grub/menu.lst
      default=0
      timeout=10
      title Potato
         root (hd0,0)
         kernel /boot/vmlinuz-2.2.19 ro root=/dev/hda1 hdc=ide-scsi
Debian$ grub-install --root-directory=/mnt/hdb1 /dev/hdb
```

Por último, desmontamos la imagen y salimos del Qemu:

```
Debian$ umount /mnt/hdb1
Debian$ halt
```

Ya podemos utilizar la nueva imagen que hemos preparado en Qemu (o en Virtualbox con el interface gráfico):

```
PC REAL$ kvm potato_con_kernel.iso -device ne2k_pci,netdev=net0 \
  -netdev user,id=net0
```

### 3.2.4. Ajustes finales

A partir de aquí solo nos queda configurar en Debian potato el teclado español, formatear el swap, cargar el modulo de red y configurar las direcciones de red:

1. Para tener teclado en español ejecutamos `kbdconfig`.

2. El swap se pone en funcionamiento con los comandos usuales de formateo (`mkswap`) y activación (`swapon`):

```
Potato$ mkswap /dev/hda2
Potato$ swapon /dev/hda2
```

3. Para que la red funcione debemos cargar el módulo del kernel del modelo de la tarjeta de red con el comando `modprobe`:

   ```
   Potato$ modprobe ne2k-pci (en KVM)
   Potato$ modprobe pcnet32 (en Virtualbox).
   ```

   La carga se hace permanente si incluimos el módulo en el fichero de carga de módulos del kernel, `/etc/modules`:

   ```
   ne2k-pci (en KVM)
   pcnet32 (en Virtualbox).
   ```

4. Para la red, los interfaces se configuran en el fichero típico de Debian, `/etc/network/interfaces`, que solo necesita dos líneas:

   ```
   iface lo inet loopback
   iface eth0 inet dhcp
   ```

   Los servidores de nombres, como es usual, se colocan en `/etc/resolv.conf`.

   NOTA: En Virtualbox adicionalmente debemos indicar el modelo *PCnet-FAST III* en las opciones avanzadas de configuración de red de la máquina.

⇒ **ENTREGA 3**: Potato con kernel completamente configurado.

## 3.3. Ejercicios no entregables

1. Crear un segundo disco duro para almacenar datos en la emulación del PC Debian sarge. Usar `dd` para crear un disco vacío de 200 MB y `fdisk` para dividirlo en dos particiones de 100 MB cada uno. Formatear la primera partición como *ext3* y la segunda como *msdos*. Arrancar Debian sarge con este segundo disco duro (*-hdb disco2.iso*) y montar las dos particiones para que los usuarios las puedan utilizar.

2. Crear una imagen de CD/DVD a partir del contenido de un directorio con el comando `genisoimage` (antiguamente `mkisofs`). Cargarlo en la distribución Debian sarge (*-cdrom cd.iso*) y comprobar que se puede leer pero no escribir.

3. Repetir el apartado 2.2 para la distribución Debian 3.0 (woody). La imagen resultante se utilizará más adelante en el capítulo de virtualización.

4. Buscar otros programas Linux que permitan crear imágenes, particionar discos o redimensionar particiones y que dispongan de interface gráfico.

# 4. Paravirtualización (Xen)

Xen es un hipervisor para la arquitectura x86, esto es, una fina capa de software que corre directamente sobre el hardware y proporciona un entorno para la ejecución y control de múltiples instancias de uno o varios sistemas operativos.

- Para procesadores x86 que no poseen extensiones de virtualización, realiza **para-virtualización**, esto es, requiere modificaciones en el kernel del sistema operativo virtual para que coopere con el kernel del sistema operativo anfitrión.

- Para los procesadores x86 con `virtualización asistida por hardware` (Intel VT o AMD-V), permite realizar virtualización completa, por lo que este caso no requiere modificaciones al kernel del sistema operativo.

En concreto, Xen permite:

- Máquinas virtuales con prestaciones próximas al hardware nativo.
- Diferentes sistemas operativos para las máquinas virtuales.
- Soporte hardware a través de los controladores de dispositivo de Linux.
- Migración en caliente de las máquinas virtuales entre anfitriones.

**Estructura**. Un sistema Xen consta de múltiples capas, siendo la más baja y privilegiada la de Xen en si mismo (el hipervisor). Xen puede albergar múltiples instancias de sistemas operativos, cada uno de los cuales se ejecuta sobre una máquina virtual segura, en lo que se denomina un dominio. Los dominios son temporizados por Xen que realiza el reparto del tiempo de CPU y del resto de los recursos hardware.

El dominio 0 (proceso `xend`) se crea automáticamente cuando el sistema arranca y tiene privilegios especiales de gestión. Este dominio construye y gestiona las máquinas virtuales. Incluye la realización de tareas tales como la suspensión, reanudación y migración de las máquinas virtuales. El usuario envía los comandos al hipervisor a través de un terminal o de uninterface HTTP.

## 4.1. Instalación

### 4.1.1. Instalación del kernel

Arrancamos el Qemu con la distribución Debian sarge y metemos dentro de la máquina virtual, con *sftp* o con un navegador, el paquete *xen-2.0.7-install-x86_32.tgz*. Descomprimimos e instalamos los archivos con:

```
$ tar -zxvf xen-2.0.7-install-x86_32.tgz
$ cd xen-2.0-install
$ sh ./install.sh
```

Con estos comandos instalamos dos kernels en el directorio *boot* y varios comandos de gestión. Los dos kernels están preparados para cooperar entre ellos:

```
/boot/vmlinuz-2.4-xen0 : kernel para el dominio 0
/boot/vmlinuz-2.4-xenU : kernel para el resto de los dominios
```

Ahora vamos a editar el fichero de arranque */boot/grub/menu.lst*. En primer lugar, modificamos la siguiente línea para indicar que por defecto arranque con el kernel número 2 de la lista:

```
default 2
```

E introducimos al final del fichero las instrucciones para arrancar los nuevos kernels (realmente solo vamos a utilizar el primero):

```
title Xen 2.0 / XenLinux 2.4
kernel /boot/xen-2.0.gz dom0_mem=131072
module /boot/vmlinuz-2.4-xen0 root=/dev/hda1 ro console=tty0

title Xen 2.0 / XenLinux 2.6
kernel /boot/xen-2.0.gz dom0_mem=131072
module /boot/vmlinuz-2.6-xen0 root=/dev/hda1 ro console=tty0
```

Por último, vamos a corregir una incompatibilidad de versiones de librerías entre Debian y el paquete binario Xen. Podemos saber las librerías que no encuentra tecleando `ldd /usr/sbin/xfrd`. Corregimos este problema con `ln`:

```
$ cd /usr/lib
$ ln -s libcurl.so.3.0.0 libcurl.so.2
$ ln -s libssl.so.0.9.7 libssl.so.4
$ ln -s libcrypto.so.0.9.7 libcrypto.so.4
```

Reiniciamos Qemu con el nuevo kernel que actuará como hipervisor. Fijarse que en menú de arranque esté seleccionado: *Xen 2.0 / XenLinux 2.4*. Entrar como el usuario *root* pues se han cambiado algunos permisos.

### 4.1.2. Creación de dominios

Iniciamos el proceso de sistema xen con el comando:

```
$ /etc/init.d/xend restart
```

Ahora necesitamos las imágenes de los dominios y sus ficheros de configuración. Vamos a probar ttylinux, que es una distribución minimalista de Linux. Introducimos dentro de la simulación Qemu el fichero *ttylinux-xen.bz2*, lo descomprimos y, como vamos a vamos a necesitar dos, sacamos una copia:

```
$ bzip2 -cd  ttylinux-xen.bz2 > ttylinux1.iso
$ cp ttylinux1.iso ttylinux2.iso
```

En */etc/xen* hay ejemplos de ficheros de configuración. Nosotros creamos uno con el nombre *ttylinux1.cfg* con el siguiente contenido:

```
kernel = "/boot/vmlinuz-2.4-xenU"
memory = 8
name = "ttylinux1"
nics = 3
vif= [ 'mac=aa:00:00:00:00:11, bridge=xen-br0', 'mac=aa:00:00:00:00:22, \
bridge=xen-br0', 'mac=aa:00:00:00:00:33, bridge=xen-br0' ]
ip = "10.0.2.50"
disk = ['file:/home/debian/ttylinux1.iso,sda1,w']
root = "/dev/sda1 ro"
```

Con *kernel* indicamos qué kernel se ejecutará al crear dominio ttylinux, en este caso uno xen, pues realmente del ttylinux sólo se utilizará el sistema de ficheros. En *memory* indicamos que el dominio ttylinux usará 8 MB de Ram. En *nics* indicamos que se usarán 3 tarjetas de red, en *vif* indicamos sus direcciones Ethernet y en *ip* la primera dirección IP. Por último, en *disk* indicamos la localización de la imagen, el dispositivo que va controlar el disco duro y los permisos, mientras que en *root* indicamos los parámetros del dispositivo raíz que se le pasará el kernel.

Creamos otro fichero *ttylinux2.cfg* con diferentes *name*, *vif*, *ip* y *disk.* Iniciamos los dominios con:

```
$ xm create ttylinux1.cfg -c
$ xm create ttylinux2.cfg -c
```

Si quisiésemos que los dominios se iniciasen automáticamente en el arranque del proceso *xend*, pondríamos los ficheros de configuración en */etc/xen/auto/*.

Podemos entrar en los dominios *ttylinux* con usuario y clave *root*. La red de los dominios está completamente operativa y podemos comprobar que:

1. Podemos hacer un *ping* desde el ttylinux1 al ttylinux2. Podemos también conectarnos desde los ttylinux al exterior, por ejemplo, ssh a la IP de la máquina real.

2. Si hemos iniciado la red de KVM con tipo *tap*, podemos hacer un ping desde el PC del laboratorio a los ttylinux. También podemos entrar al ttylinux desde el PC del laboratorio como usuario y clave *guest*.

Podemos acabar con el dominio desde dentro con `halt`.

### 4.1.3. Gestión de dominios

Disponemos de los siguientes comandos para operar sobre las máquinas virtuales desde el hipervisor:

1. Creación, apagado y destrucción de dominios (máquinas virtuales). Con la opción `-c` indicamos que al arrancar entre en una consola. La máquina virtual también puede apagarse desde dentro con `halt` (o `poweroff`).

   ```
   $ xm create dominio.cfg -c
   $ xm shutdown dominio
   $ xm destroy dominio
   ```

2. Ayuda y lista de máquinas virtuales:

   ```
   $ xm help
   $ xm list
   $ xm list -l
   ```

   La salida del comando `xm list` da la siguiente información:

   ```
   name domid memory cpu state cputime console
   ```

   Los dos primeros campos nos dicen el nombre y el ID que tiene asignado el dominio. A continuación viene la memoria y qué cpu está utilizando. El estado puede ser: *r* running, *b* blocked, *p* paused, *s* shutdown y *c* crashed. Luego viene el tiempo de cpu usado. Por último, el número de consola asignada.

3. Acceso a los dominios. Dado un número de consola podemos entrar en el dominio (con la condición de que no hayamos entrado antes, porque sólo permite una conexión por consola).

   ```
   $ xm consoles : lista las consolas asociadas a los dominios
   $ xm console dominio : crea una nueva consola para un dominio
   $ xm console id : crea una nueva consola para un dominio
   $ xencons localhost consola
   $ telnet localhost consola
   ```

4. Parada, reanudación y grabación de estado en fichero (se guarda en el fichero *dominio.xen*):

   ```
   $ xm pause dominio
   $ xm unpause dominio
   $ xm save dominio dominio.xen
   $ xm restore dominio.xen
   ```

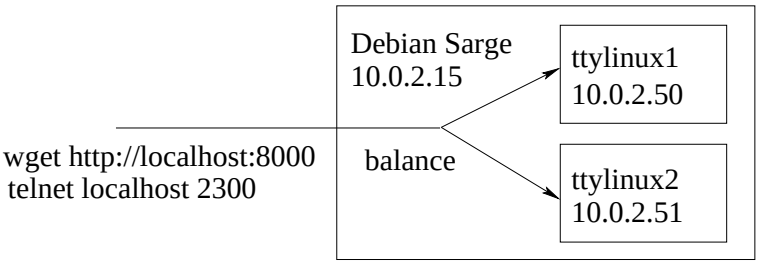
5. Migración. Los dominios pueden transferirse de un servidor a otro sin necesidad de pararlos (lo que se denomina migración). Existe un comando que permite hacer esto automáticamente:[10]

```
$ xm migrate --live dominio servidor_destino
```

6. La memoria máxima a usar por el dominio puede ajustarse con:

```
$ xm maxmem dominio MB
```

⇒ **ENTREGA 4:** Balanceo de los accesos a dos servidores ttylinux virtuales para los servicios http y telnet.




Descargar en Debian sarge un programa simple de balanceo, por ejemplo, *balance-3.57.tar.gz*,[11] que es un programa de sockets que actúa como proxy. Para instalar el programa, descomprimir el archivo, entrar en el directorio, teclear *make* y *make install*. El balanceo lo hacemos con los comandos:

```
Qemu$ balance -b 10.0.2.15 80 10.0.2.50 10.0.2.51
Qemu$ balance -b 10.0.2.15 2300 10.0.2.50:23 10.0.2.51:23
```

Para que los accesos sean diferenciables vamos a modificar ligeramente las páginas web que proporcionan los ttylinux. Puesto que los ttylinux no tienen editores de texto, vamos a descargar la página web, editarla fuera y reintegrarla modificada al servidor ttylinux:

```
Qemu$ tiny.local login: (root, root)
TTYLinux$ cd /home/web
TTYLinux$ ftp 10.0.2.15 (debian, debian)
ftp> put index.html
---> Modificamos ligeramente index.html en Debian sarge
ftp> get index.html
ftp> quit
```

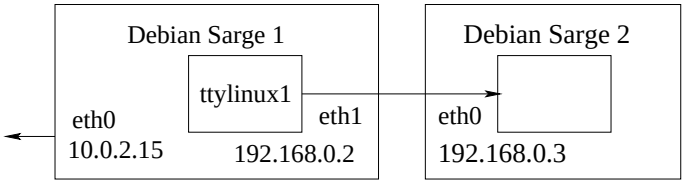
[10] Comprobar que está corriendo el proceso `xfrd` con `cat /var/run/xfrd.pid`.
[11] Disponible en *https://www.inlab.de/balance.html*.

Probamos los servicios desde el PC del laboratorio o desde un portátil:

```
PC REAL$ unset http_proxy (si estamos en un PC de cable del laboratorio)
PC REAL$ wget http://localhost:8000 (leer el contenido)
PC REAL$ wget http://localhost:8000 (leer el contenido)

PC REAL$ telnet localhost 2300 (guest, guest)
TTYLinux$ ifconfig eth0
TTYLinux$ exit
```

⇒ **ENTREGA 5:** Migración de un *ttylinux* entre dos Qemu que ejecutan Xen.




Realizar los siguientes pasos:

1. Partir de una instalación Xen con ttylinux virtualizado funcionando.

2. Apagar Debian sarge y sacarle una copia (`cp debian1.img debian2.img`) para KVM o clonarlo en Virtualbox. Es importante que las dos copias tengan las imágenes ttylinux y su configuración.

3. El primer anfitrión tendrá dos interfaces red y el segundo uno. Vamos a crear una red interna que conecte el interface *eth1* del primero con el *eth0* del segundo. Para ello, arrancar los dos anfitriones con el script `inicia_dos_kvm.sh` (KVM) o definir en la configuración de red de las máquinas una *internal network* (Virtualbox).

   Una vez arrancadas las máquinas virtuales, configurar en la primera los dos interfaces con el comando `ifconfig` (es lo más sencillo y rápido, aunque no es permanente) o editando los ficheros `/etc/network/interfaces` y en la segunda el único interface de red.

   Antes de seguir, comprobar que tenemos conexión haciendo un ping de 192.168.0.2 a 192.168.0.3 y viceversa.

4. Arrancar Xen en los dos anfitriones y arrancar ttylinux en el primero.

5. Migrar ttylinux al segundo Qemu.

## 4.2. Ejercicios no entregables

1. Virtualización en Xen de la distribución Debian 2.2 Potato. Crear un dominio con la imagen iso de la distribución Debian 2.2 Potato (sin kernel) elaborada en el capítulo anterior con el comando `debootstrap` y que tendremos dentro del Qemu.
2. Usar un segundo disco duro (*-hdb disco2.iso)* y descomprimir en una partición los ficheros de la distribución Debian 3.0 woody elaborada en el capítulo anterior con el comando `debootstrap`. En el fichero de configuración de Xen especificar:

   ```
   disk = ['phy:hdb1,sda1,w']
   ```

   Avanzado: exportar por NFS la partición y hacer que Xen acceda por red a la imagen (véase el capítulo 5.4 del manual de usuario Xen 2.0).
3. Instalar la versión 3.0.2 de Xen (no es necesario borrar la anterior). Descomprimiendo el archivo se instala el nuevo kernel, los módulos y los comandos. Conviene generar las dependencias de los módulos del nuevo kernel y crear una imagen de arranque:

   ```
   $ cd /lib/modules/2.6.16-xen
   $ depmod -v 2.6.16-xen  (se generan modules.*)
   $ mkinitrd -o  /boot/initrd-2.6-xen.img 2.6.16-xen
   ```

   Hay que añadir las siguientes líneas al menú de arranque */boot/grub/menu.lst*:

   ```
   title Xen 3.0 / XenLinux 2.6
   kernel /boot/xen-3.0.gz console=vga
   module /boot/vmlinuz-2.6-xen root=/dev/hda1 ro console=tty0
   module /boot/initrd-2.6-xen.img
   ```

   Sólo queda reiniciar el sistema seleccionando la opción *Xen 3.0 / XenLinux 2.6* del menú de arranque. Probar la nueva versión de xen con el ttylinux.

# 5. Contenedores (OpenVZ)

**NOTA IMPORTANTE:** Esta práctica ha sido sustituida por la del siguiente apartado (LXC y Docker), que es más moderna, pero requiere un PC más potente. Por tanto, este guión se deja aquí por si en algún portátil no se puede ejecutar la siguiente práctica.

OpenVZ realiza virtualización de sistema operativo o de contenedores, es decir, se ejecuta por encima del sistema operativo de la máquina, proporcionado a los servidores virtuales (denominados aquí contenedores) un conjunto de librerías con las que pueden interactuar. Estas librerías traducen las llamadas que realizan los servidores virtuales de modo que puedan ser ejecutadas por el sistema operativo que corre sobre el hardware. De esta forma, los servidores virtuales quedan aislados en los contenedores. Cada contenedor posee:

- Sus propios procesos, ficheros, usuarios, etc.
- Sus propia configuración de red, rutado, filtrado, etc.
- Su propia versión de librerías, aplicaciones, etc.

## 5.1. Instalación

### 5.1.1. Instalación del kernel y las librerías

Arrancamos Qemu con Debian sarge y nos traemos, por ejemplo con *sftp* o con un navegador, los ficheros de la práctica *ovzkernel-2.6.16.tar.gz*, *vzctl-3.0.23-1.i386.tar.gz*, *vzctl-lib-3.0.23-1.i386.tar.gz* y *vzquota-3.0.12-1.i386.tar.gz*. Descomprimimos e instalamos los archivos con:

```
$ cd /
$ tar -zxvf /home/debian/ovzkernel-2.6.16.tar.gz
$ tar -zxvf /home/debian/vzctl-3.0.23-1.i386.tar.gz
$ tar -zxvf /home/debian/vzctl-lib-3.0.23-1.i386.tar.gz
$ tar -zxvf /home/debian/vzquota-3.0.12-1.i386.tar.gz
$ chmod 755 /etc /usr /usr/sbin /usr/lib /usr/share /var /var/lib
$ chmod 755 /usr/share/man /usr/share/man/man5 /usr/share/man/man8
```

Con estos comandos instalamos un nuevo kernel, las librerías y algunos comandos como `vzctl`. El nuevo kernel está preparado para atender tanto a las llamadas del huésped como de las máquinas virtuales. El comando `chmod` lo usamos para restaurar los permisos que ha modificado la instalación, pues de otro modo los usuarios no privilegiados no podrían entrar en el sistema.

A continuación, editamos el fichero de arranque */boot/grub/menu.lst*. En primer lugar, modificamos la siguiente línea para indicar que por defecto arranque con el kernel número 4 de la lista (o número 2 si no tuviéramos las líneas de Xen):

```
default 4
```

E introducimos al final del fichero las instrucciones para arrancar este kernel:

```
title  OVZ 2.16.16
root   (hd0,0)
kernel /boot/vmlinuz-2.6.16-026test020.1 root=/dev/hda1 ro
```

A continuación, configuramos las opciones de red en el kernel incluyendo en el fichero */etc/sysctl.conf* las siguientes líneas (de las cuales *ip_forward* es la conocida retransmisión de paquetes):

```
# packet forwarding enabled and proxy arp disabled
net.ipv4.ip_forward = 1
net.ipv4.conf.default.proxy_arp = 0
# Enables source route verification
net.ipv4.conf.all.rp_filter = 1
# Enables the magic-sysrq key
kernel.sysrq = 1
# we do not want all our interfaces to send redirects
net.ipv4.conf.default.send_redirects = 1
net.ipv4.conf.all.send_redirects = 0
```

Reiniciamos con `reboot` y fijarse que en menú de arranque esté seleccionado *OVZ 2.16.16*.

### 5.1.2. Creación de contenedores

Iniciamos el proceso de sistema openvz con el comando:

```
$ /etc/init.d/vz restart
```

Ahora necesitamos la imagen de los contenedores y sus ficheros de configuración. Vamos a probar *woody* que es una versión antigua (v3.0) de Debian.[12] Introducimos dentro de la simulación Qemu el fichero *woody.tar.bz2* (que hemos elaborado con el comando debootstrap) y lo descomprimimos:

```
$ cd /vz/private
$ tar -jxvf /home/debian/woody.tar.bz2
```

Tenemos que crear una configuración para el contenedor. Elegimos un identificador, por ejemplo, 555, pero antes comprobamos que no está siendo usado por otro contenedor con el comando:

[12] Ttylinux no es compatible con OpenVZ, la versión Debian 2.2 (potato) es demasiado antigua y no reconoce la tarjeta de red. La versión 3.3 (sarge) sería más adecuada pero ocupa más espacio.

```
$ vzlist -a 555
```

Si la respuesta es *Container(s) not found* lo configuramos:

```
$ mv woody 555
$ vzctl set 555 --applyconfig vps.basic --save
$ echo "OSTEMPLATE=debian-3.0" >> /etc/vz/conf/555.conf
```

La configuración de red del contenedor la realizamos de la siguiente forma:

```
$ vzctl set 555 --hostname pepito.com --save
$ vzctl set 555 --ipadd 192.168.0.10 --save
$ vzctl set 555 --nameserver 193.144.75.9 --save
```

Podemos comprobar que la configuración del contenedor se ha realizado correctamente consultando el fichero */etc/vz/conf/555.conf* (líneas del final). Una vez creado el contenedor no debemos tocar su sistema de ficheros (*/vz/private/555*), sino que cualquier modificación se realizará desde el propio contenedor o con los comandos apropiados. Iniciamos el contenedor y entramos en el, respectivamente,

```
$ vzctl start 555
$ vzctl enter 555
```

Podemos fijar la clave para el usuario root en el contenedor. Y si queremos algún programa adicional, por ejemplo, `ssh` podemos instalarlo:

```
pepito$ passwd
pepito$ apt-get install ssh
```

Salimos del contenedor con `exit`. Comprobamos que funciona la red desde Debian sarge con `ping 192.168.0.10`. Ahora ya podemos entrar en el contendor con `ssh root@192.168.0.10`.

### 5.1.3. Gestión de contenedores

1. La creación y destrucción de contenedores se realiza con los comandos siguientes (antes de destruir un contenedor tenemos que pararlo):[13]

   ```
   $ vzctl
   $ vzctl start ID
   $ vzctl destroy ID   # OJO!!! NO ES RECUPERABLE
   ```

[13] Luego quedan por borrar a mano */etc/vz/conf/555.conf.destroyed* y */vz/vztmp/**.

2. Podemos listar, parar, reanudar o comprobar el estado de un contenedor con los siguientes comandos (también puede salirse desde dentro con `poweroff`):

```
$ vzlist -a
$ vzctl stop ID --fast
$ vzctl restart ID
$ vzctl status ID
```

3. Realmente casi nunca tendremos que entrar en un contenedor, pues podemos ejecutar remotamente los comandos. Para ejecutar un comando y para cambiar la clave de un usuario, tenemos:

```
$ vzctl exec ID comando
$ vzctl set ID --userpasswd usuario:clave
```

4. Una vez que estamos seguros de que el contenedor funciona (y esté apagado), podemos guardar una plantilla para la creación de más instancias de contenedores:

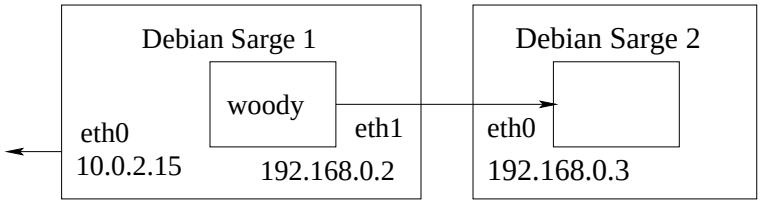
```
$ cd /vz/private/555
$ tar -czvf /vz/template/cache/plantilla-woody.tar.gz .
```

5. Las instancias de nuevos contenedores se crearán a partir de la plantilla de la forma:

```
$ vzctl create 666 --ostemplate plantilla-woody
$ vzctl set 666 --ipadd 192.168.0.11 --save
...
$ vzctl start 666
```

### 5.1.4. Migración de contenedores

Los contenedores pueden transferirse de una máquina a otra sin necesidad de pararlos (lo que se denomina migración). Supondremos el siguiente esquema:



1. Partir de una instalación Debian sarge con Openvz funcionando.
2. Apagar Debian sarge y sacarle una copia (`cp debian1.img debian2.img`) para KVM o clonarlo en Virtualbox.
3. El primer anfitrión tendrá dos interfaces red y el segundo uno. Vamos a crear una red interna que conecte el interface *eth1* del primero con el *eth0* del segundo. Para ello, arrancar los dos anfitriones con el script `inicia_dos_kvm.sh` (KVM) o definir en la configuración de red de las máquinas una *internal network* (Virtualbox).

   Una vez arrancadas las máquinas virtuales, configurar en la primera los dos interfaces con el comando `ifconfig` (es lo más sencillo y rápido, aunque no es permanente) o editando los ficheros `/etc/network/interfaces` y en la segunda el único interface de red.

   Antes de seguir, comprobar que tenemos conexión haciendo un ping de 192.168.0.2 a 192.168.0.3 y viceversa.
4. Uno de los anfitriones ejecutará el contenedor 555 y el otro no.

Antes de transferir el contenedor ha de compartirse una clave para el comando `ssh root@destino` se realize sin preguntar por clave. Para ello, la clave pública *ssh* del origen debe ser colocada en el destino en el fichero */root/.ssh/authorized_keys* lo que puede realizarse con el script que se proporciona a tal efecto:

```
DebianOrigen$ sh /usr/local/bin/genera_public_key.sh root@IP_destino
DebianOrigen$ vzmigrate --online IP_destino ID
```

⇒ **ENTREGA 6**. Migrar un contenedor entre dos hosts con OpenVZ.

**NOTA IMPORTANTE:** Esta entrega 6 solo se realizará en caso de que por problemas en la potencia del portátil no se pueda realizar la siguiente práctica.

## 5.2. Ejercicios no entregables

1. Monitorizar el consumo de CPU y de memoria (véase manual de usuario).
2. Establecer una cuota de disco para un contenedor (véase manual).

# 6. Contenedores (LXC y Docker)

**NOTA IMPORTANTE:** En caso de no poder realizar esta prácticas por la capacidad del portátil se realizará la entrega 6 del anterior apartado.

## 6.1. Configuración de la máquina virtual

En las siguientes prácticas utilizaremos una imagen Debian más reciente, lo que nos permite trabajar con los últimos virtualizadores, aunque con la desventaja de ocupar mucho más espacio, ejecución mucho más lenta y con instalaciones que requieren descargar varios GB de datos. También utilizaremos `virtualbox` en lugar de QEMU/KVM, aunque podría utilizarse sin problemas este último. Por último, utilizaremos volúmenes lógicos para almacenar las imágenes virtuales.

En concreto, utilizaremos una máquina virtual Debian *buster* (versión 10.2) de 64 bits con las particiones de la siguiente tabla.

| Partición | Tamaño | Tipo | Uso |
|---|---|---|---|
| 1 | 512 MB | ext4 | /boot |
| 2 | 20 GB | ext4 | / |
| 3 | 4 GB | swap | Área de intercambio |
| 4 | resto (∼ 61.4 GB)<br>inicialmente vacío<br>por configurar | LVM | Imágenes LXC (20 GB)<br>Imágenes Docker (20 GB)<br>Otras imágenes (20 GB) |

El disco que hemos preparado tiene 80 GB y contiene 4 particiones, de los cuales algo más de 60 GB se dedicarán a almacenar las máquinas virtuales mediante el gestor de volúmenes lógicos LVM. Es decir, que cada vez que se necesite una máquina virtual se creará un volumen lógico para contenerlo.

La imagen proporcionada es `aasr1.ova`, preparada para importar en el programa `virtualbox`. El formato OVA es una versión comprimida con TAR del *Open Virtualization Format (OVF)*, un estándar abierto (de intercambio) para empaquetar y distribuir servicios virtualizados.

Para poder acceder por ssh y a los servicios telnet y http, en virtualbox debemos teclear en un terminal del PC o del portátil, lo siguiente:

```
VBoxManage modifyvm aasr1 --natpf1 "guestssh,tcp,127.0.0.1,2222,,22"
VBoxManage modifyvm aasr1 --natpf1 "guesttelnet,tcp,127.0.0.1,2300,,2300"
VBoxManage modifyvm aasr1 --natpf1 "guesthttp,tcp,127.0.0.1,8000,,80"
VBoxManage modifyvm aasr1 --natpf1 "guestvnc,tcp,127.0.0.1,5900,,5900"
```

Una vez arrancada la imagen en *virtualbox* disponemos de los usuarios *debian* y *root*, ambos con clave *debian*, y el editor *nedit*. Como siempre, podemos conectarnos con `ssh -p 2222 debian@localhost` y luego `su -` para facilitar la entrada de comandos. Lo primero que debemos hacer es configurar LVM, lo que se hace con los siguientes comandos:

1. Actualizar la lista de paquetes: `apt-get update`
2. Instalar el paquete de LVM: `apt-get install lvm2`
3. Crear con el comando `fdisk /dev/sda` la partición primaria 4 para LVM que ocupará totalmente el espacio vacío del disco. A la partición creada le podemos fijar el identificador apropiado (*Linux LVM*) con la opción *t: change a partition type*.
4. Preparar un volumen físico con la partición creada (`pvcreate`) y crear el grupo de volúmenes (`vgcreate`). Podemos comprobar las particiones y volúmenes con los comandos `lsblk` y `vgs`:

```
$ pvcreate /dev/sda4
$ vgcreate nombre_apellido1_apellido2  /dev/sda4
$ lsblk
$ vgs
```

**IMPORTANTE:** el nombre y apellidos se utilizará para comprobar que cada alumno trabaja individualmente.

**debootstrap y chroot**

- `chroot` fue una de las primeras aplicaciones que proporcionó aislamiento de procesos, por lo que es el antecesor de los programas de contenedores modernos. Sin embargo es demasiado simple y le faltan muchas características que tienen actualmente los contenedores. El primer sistema de contenedores propiamente dicho introducido en Linux fue OpenVZ.
- Por su parte, `debootstrap` se utiliza para descargar de los servidores de la distribución Debian una versión del S.O. apropiada para ejecutar con chroot.

A modo de ejemplo, podemos crear un volumen lógico de 200 MB para la imagen virtual que se genera con `debootstrap` y `chroot`, siguiendo los pasos vimos en el apartado de creación de máquinas virtuales:

```
$ lvcreate -L200M nombre_apellido1_apellido2 -n potato1
$ mkfs.ext4 /dev/mapper/nombre_apellido1_apellido2-potato1
$ mkdir /home/debian/potato1
$ mount /dev/mapper/nombre_apellido1_apellido2-potato1 /home/debian/potato1
$ debootstrap --arch i386 --no-check-gpg potato /home/debian/potato1 \
   http://archive.debian.org/debian
$ chroot /home/debian/potato1
```

El resto seguiría como vimos en el correspondiente apartado.

## 6.2. Contenedores Linux

LXC (Linux Containers) es una tecnología de virtualización a nivel de sistema operativo o de contenedor propia de Linux. Es un sistema de contenedor moderno y ligero que realiza el aislamiento de procesos a la vez que proporciona acceso al sistema operativo. En la virtualización de sistema operativo o de contenedores, las máquinas virtuales no incluyen su propio kernel, pues ejecutan el del host.

LXC es el sucesor de OpenVZ, pues al principio los programadores de OpenVZ modificaban el kernel de Linux para incluir en el las características de contenerización, pero pasado el tiempo estas modificaciones se incluyeron de serie en el kernel y LXC se diseñó para utilizarlas.

La instalación de LXC se realiza de la siguiente forma:

1. Instalamos el paquete de LXC con: `apt-get install lxc`
2. El servicio se denomina `lxc`. Por ejemplo:

   ```
   $ systemctl status lxc
   $ systemctl restart lxc
   ```

3. Descargamos una imagen previamente preparada por la distribución, la vamos a denominar *contenedor1* y crearemos un volumen lógico del mismo nombre para contenerla. Estos dos pasos lo realiza el comando:

   ```
   $ lxc-create -t download -n contenedor1 --bdev lvm --vgname \
      nombre_apellido1_apellido2 --lvname contenedor1
   ```

   Nos da una lista de distribuciones y seleccionaremos una, por ejemplo:

   ```
   Distribution: voidlinux
   Release: current
   Architecture: i386
   ```

4. Los comandos de gestión de los contenedores son los típicos:

| Comando | Descripción |
|---|---|
| `lxc-ls -f` | Lista los contenedores que tenemos configurados, `-f` da el estado de los contenedores |
| `lxc-start -n` *`contenedor1`* | Iniciamos un contenedor |
| `lxc-info -n` *`contenedor1`* | Nos da el estado y consumo de un contenedor |
| `lxc-attach -n` *`contenedor1`* | Entramos en el contenedor |
| `lxc-attach -n` *`contenedor1 comando`* | Ejecutamos un comando en el contenedor |
| `lxc-stop -n` *`contenedor1`* | Paramos la ejecución del contenedor |
| `lxc-destroy -n` *`contenedor1`* | Borramos todo del contenedor |

## 6.3. libvirt

**libvirt** es un entorno de virtualización que incluye una API de código abierto, un proceso de sistema (demonio o servicio) y una herramienta de administración para gestionar la virtualización en Linux. Se puede usar para administrar KVM, QEMU, Xen, LXC, OpenVZ, VMware y otras tecnologías de virtualización.

1. Se instala con `apt-get install libvirt-daemon-system virt-manager` (la aplicación y un gestor gráfico).

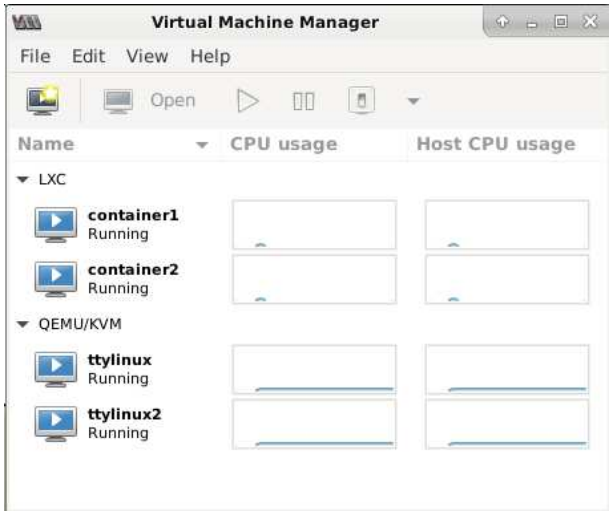
2. El servicio es `libvirtd`, con el que podemos trabajar de forma usual, por ejemplo:

```
$ systemctl restart libvirtd
$ systemctl status libvirtd
$ systemctl stop libvirtd
```

3. Podemos trabajar en un terminal para configurar las máquinas virtuales con el comando `virsh`, pero es más cómodo trabajar con una interface grafica, por ejemplo, `virt-manager`. Para configurar una máquina virtual, se requieren básicamente dos pasos:

   a) *Añadir una conexión*. Por defecto conecta con QEMU/KVM, pero podemos decirle, por ejemplo, que conecte con LXC.

   b) *Nueva máquina virtual*. Le indicamos la conexión y el nombre de la máquina virtual en el correspondiente virtualizador. Se conecta con el virtualizador y obtiene los datos de la máquina virtual. Y a partir de aquí podemos gestionar la máquina virtual con libvirt,




⇒ **ENTREGA 6.a**: Un contenedor LXC gestionado por virt-manager.

## 6.4. Docker

Docker, al igual que LXC, es un virtualizador de sistema operativo o de contenedores, pero adopta un enfoque muy diferente. En lugar de ejecutar un contenedor similar a una máquina virtual con una pila de software completa (sistema init, servicios, etc) y todas las demás cosas que un S.O. pueda tener, está diseñado para ejecutar una sola aplicación. Tanto Docker como LXC usan las características de contenerización del kernel de Linux, pero están implementados de forma independiente.

Docker utiliza un fichero denominado `Dockerfile`, que incluye toda la información necesaria para crear un entorno en el que se pueda ejecutar la aplicación. Docker usa este Dockerfile para construir una imagen de la aplicación (básicamente un conjunto de ficheros TAR) y las bibliotecas que necesita, a continuación se crea una imagen y finalmente se ejecuta a través de las características de contenerización del kernel de Linux. Todo este proceso es muy rápido, del orden de milisegundos.

Cada imagen de Docker se compone de una serie de capas. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un comando, como ejecutar o copiar, se crea una nueva capa. Estas capas se combinan en una sola imagen. Docker reutiliza estas capas para construir nuevos contenedores, lo cual hace mucho más rápido el proceso de construcción. Los cambios intermedios se comparten entre imágenes, mejorando aún más la velocidad, el tamaño y la eficiencia.[14]

### 6.4.1. Instalación de Docker básica

Para la instalación de docker en nuestra máquina virtual Debian utilizaremos los siguientes pasos:[15]

1. Si estamos en un PC de cable de la USC debemos configurar el proxy. Conviene incluir la configuración en `.bashrc` (de root y del usuario) para que se cargue automáticamente al abrir un terminal:

   ```
   $ cat <<EOF | tee -a /root/.bashrc
   export http_proxy="http://proxy2.usc.es:8080"
   export https_proxy="http://proxy2.usc.es:8080"
   export ftp_proxy="http://proxy2.usc.es:8080"
   EOF
   ```

   NOTA: El comando `tee` escribe en salida estándar y en fichero (en ambos a la vez) y la opción `-a` indica añadir. Toma como entrada el texto que le

[14] https://www.redhat.com/es/topics/containers/what-is-docker

[15] Adaptado de http://www.itzgeek.com/how-tos/linux/debian/how-to-install-docker-on-debian-9.html y https://blog.geruechtekueche.de/docker-under-ubuntu-16-04-set-up-for-direct-lvm-storage/.

proporciona el comando `cat`, en donde la construcción `cat << ETIQUETA` *líneas* `ETIQUETA` sirve para indicarle el comienzo y el fin del texto al comando `cat`.

Más adelante, en el apartado de gestión de contenedores, tendremos que configurar el servicio Docker para que conecte con el servidor de contenedores usando el proxy:

```
$ mkdir -p /etc/systemd/system/docker.service.d

$ cat <<EOF | tee -a /etc/systemd/system/docker.service.d/https-proxy.conf
[Service]
Environment="https_proxy=https://proxy2.usc.es:8080/"
EOF

$ systemctl daemon-reload
$ systemctl restart docker
```

2. Vamos a crear un volumen lógico para almacenar todos los ficheros de Docker. No es necesario dedicar un volumen lógico a cada máquina virtual pues Docker ya se encarga el mismo de gestionar el espacio apropiadamente.

   ```
   $ lvcreate -L20G nombre_apellido1_apellido2 -n docker
   $ mkfs.ext4 /dev/mapper/nombre_apellido1_apellido2-docker
   $ mkdir /var/lib/docker/
   $ mount /dev/mapper/nombre_apellido1_apellido2-docker /var/lib/docker/
   ```

   Para no tener que volver a montar el volumen lógico cada vez que arrancamos, lo añadimos a `/etc/fstab`:

   ```
   $ cat <<EOF | tee -a /etc/fstab
   /dev/mapper/nombre_apellido1_apellido2-docker /var/lib/docker/ \
      ext4 defaults 0 2
   EOF
   ```

3. Instalamos desde nuestra distribución los siguientes paquetes genéricos:

   ```
   $ apt-get update
   $ apt-get install -y apt-transport-https ca-certificates \
      wget software-properties-common
   ```

4. La instalación hay que realizarla desde el sitio oficial de Docker, por lo que añadimos el servidor de paquetes de Docker a la lista de fuentes de APT. Necesitamos la clave del servidor docker y utilizaremos un fichero en el directorio `/etc/apt/sources.list.d/` para contener el URI (se podría utilizar alternativamente el fichero de configuración `/etc/apt/sources.list`).

```
$ wget https://download.docker.com/linux/debian/gpg
$ apt-key add gpg

$ cat <<EOF | tee -a /etc/apt/sources.list.d/docker.list
deb [arch=amd64] https://download.docker.com/linux/debian \
   $(lsb_release -cs) stable
EOF
```

5. Añadido el nuevo servidor, actualizamos la lista de paquetes, comprobamos las políticas de APT e instalamos el paquete que contiene docker (`docker-ce`):

   ```
   $ apt-get update
   $ apt-cache policy docker-ce
   $ apt-get install docker-ce
   ```

   Notar que durante la instalación de paquetes realizada por el último comando se instala un nuevo kernel, que está preparado para atender tanto a las llamadas del huésped como de los contenedores.

6. Comprobamos que el paquete docker se instaló correctamente comprobando el servicio `docker`:

   ```
   $ systemctl restart docker
   $ systemctl status docker
   ```

### 6.4.2. Gestión de contenedores

Instalado Docker, ya podemos iniciar nuestro primer contenedor:

```
$ docker run hello-world
```

La salida nos dice que el contenedor no está almacenado localmente, por lo que primero se descarga y luego se ejecuta:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
...
Status: Downloaded newer image for hello-world:latest
...
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Los comandos de gestión de los contenedores son los usuales:

1. Buscar imágenes en el repositorio de Docker usando una palabra clave, por ejemplo, debian:

```
$ docker search debian
   NAME    DESCRIPTION                  STARS  OFFICIAL  ...
   ubuntu  Ubuntu is a Debian-based... 10301  [OK]
   debian  Debian is a Linux ...         3313  [OK]
```

2. Traer una imagen del repositorio, por ejemplo, ubuntu:15.04,

```
$ docker pull ubuntu:15.04
```

3. Listar las imagenes almacenadas localmente:

```
$ docker images
   REPOSITORY   TAG     IMAGE ID      CREATED        SIZE
   hello-world  latest  fce289e99eb9  12 months ago  1.84kB
   debian       latest  67e34c1c9477  5 weeks ago    114MB
```

4. Ejecutar una image (`-it` indica interactivo en terminal y `-d` indica *detached mode*, esto es, finaliza cuando termina el proceso):

```
$ docker run -it -d <image-name>
```

5. Ejecuta un comando, por ejemplo *bash* (necesita el container-id):

```
$ docker exec -it ec10757e8300 bash
root@ec10757e8300:/# ls
   bin  dev home lib64 mnt proc run  srv tmp var
   boot etc lib  media opt root sbin sys usr
```

6. Listar los contenedores en ejecución (`docker ps`) o listar los contenedores parados y en ejecucion (`docker ps -a`)

```
$ docker ps -a
   CONTAINER ID  IMAGE        COMMAND  CREATED        STATUS       ...
   8c2a726709f7  hello-world  "/hello" 3 minutes ago  Exited (0)
   ec10757e8300  debian       "bash"   2 minutes ago  Up 2 minutes
```

7. Parar un contenedor, borrar un contenedor parado y borrar una imagen almacenada localmente:

```
$ docker stop <container-id>
$ docker rm <container-id>
$ docker rmi <image-id>
```

8. Para borrar todos los contenedores que estén parados y sus correspondientes imágenes:

```
$ docker container prune
$ docker system prune
```

### 6.4.3. Dockerfiles y conexión VNC con los contenedores

Para la conexión por VNC (*Virtual Network Computing*) con los contenedores instalamos en nuestro portátil un cliente de VNC, de los muchos que hay disponibles: `xtightvncviewer`, `krdc`, `xvnc4viewer`, `remmina`, etc. Instalamos el que más nos guste. Para probarlo, podemos ejecutar un contenedor Docker ya preparado que lanza *firefox*:

```
$ docker run -p 5900:5900 -e HOME=/ firefox-vnc:latest x11vnc \
   -forever -usepw -create
```

Nos conectamos desde nuestro portátil vía VNC. Los parámetros que necesitamos son:

```
VNC server: 127.0.0.1:5900
Password: 1234
```

A continuación vamos a elaborar nosotros mismos el contenedor que lanza *firefox* a partir de un *Dockerfile*. Para ello creamos el fichero `firefox-vnc`:

```
# Firefox sobre VNC
FROM    ubuntu
# Actualizamos e instalamos firefox y vnc, xvfb para el display
RUN     apt-get update
RUN     apt-get install -y firefox x11vnc xvfb
# Ponemos a clave "debian" para vnc
RUN     mkdir /.vnc
RUN     x11vnc -storepasswd debian /.vnc/passwd
# Preparamos .bashrc para que arranque firefox
RUN     bash -c 'echo "firefox" >> /.bashrc'
```

Como se puede ver, la creación de un Dockerfile es muy sencilla. Primero con la sentencia `FROM` indicamos la imagen de la que vamos a partir y a continuación con sentencias `RUN` vamos ejecutando comandos para configurar la imagen. Una vez escrito el fichero, lo compilamos (no olvidarse del punto del final):

```
$ docker build -t firefox-vnc:v1 --file ./firefox-vnc .
```

Finalmente ejecutamos la imagen con `docker` indicando el puerto 5900 para VNC y el directorio $HOME. El comando que indicamos en la sentencia es realmente `x11vnc -forever -usepw -create` (arrancar el servidor VNC con clave). La ventana de `firefox` se abre al leer bash el fichero `.bashrc`.

```
$ docker run -p 5900:5900 -e HOME=/ firefox-vnc:v1 \
   x11vnc -forever -usepw -create
```

⇒ **ENTREGA 6.b:** Elaborar un Dockerfile de solo 4 líneas que además de *firefox* abra tambien *nedit*, partiendo de la imagen `firefox-vnc` que ya tenemos almacenada localmente. La ventana de *nedit* aparecerá cuando se cierre firefox.

## 6.5. Kubernetes

Existen dos herramientas para gestionar granjas de ordenadores corriendo contenedores Docker: *Kubernetes* y *Docker Swarms*. Ambas distribuyen la carga de los contenedores entre los hosts de manera balanceada, segura y tolerante a fallos. **Kubernetes**, desarrollado por Google, es más complejo y flexible, por lo que puede gestionar cualquier situación. **Docker Swarms**, por su parte, es más sencillo y se gestiona con los mismos comandos que se usan para construir los contenedores Docker. En este apartado veremos como realizar la configuración de *Kubernetes* usando la utilidad *Minikube*, que permite gestionar múltiples contenedores dentro de un mismo host. Lo haremos para balancear los accesos a contenedores con servidores web.

Importante: Minikube necesita por lo menos dos núcleos, por lo que tenemos configurar en virtualbox la máquina virtual aasr1 para que tenga dos núcleos.

1. Instalamos el repositorio:

```
$ apt-get update
$ wget https://packages.cloud.google.com/apt/doc/apt-key.gpg
$ apt-key add apt-key.gpg

$ cat <<EOF | tee -a /etc/apt/sources.list.d/kubernetes.list
deb [arch=amd64] https://apt.kubernetes.io/ kubernetes-xenial \
   main /etc/apt/sources.list.d/kubernetes.list
EOF
```

   Y a continuación instalamos los paquetes:

```
$ apt-get update
$ apt-get install kubectl kubeadm kubelet
```

2. También descargamos e instalamos *Minikube*:

```
$ wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
$ chmod +x minikube-linux-amd64
$ mv minikube-linux-amd64 /usr/local/bin/minikube
```

3. Ahora arrancamos la aplicación para crear el clúster. Como no tenemos virtualización (Minikube puede trabajar con KVM o virtualbox), indicamos como controlador *none* (tarda un rato en ejecutar):

```
$ minikube start --vm-driver=none
```

4. Vamos a crear un conjunto de contenedores que atiendan peticiones web. Para ello preparamos los dos siguientes ficheros, página web y Dockerfile:

```
$ mkdir html
$ cat <<EOF | tee -a html/index.html
<HTML>
<HEAD> <TITLE>Mi pagina web</TITLE> </HEAD>
<BODY> HOLA </BODY>
<HTML>
EOF

$ cat <<EOF | tee -a cat hola-minikube
FROM nginx
COPY html /usr/share/nginx/html
EOF
```

5. Compilamos el Dockerfile y ejecutamos kubernetes:

```
$ docker build -t hola-minikube:v1 --file ./hola-minikube .
$ kubectl run hola-minikube --image hola-minikube:v1 --port=80
$ kubectl get deployments
$ kubectl expose deployment hola-minikub --type=LoadBalancer
$ kubectl get services
$ minikube service hola-minikube
```

   Nos da la siguiente salida:

```
|-----------|---------------|-------------|-----------------------|
| NAMESPACE |      NAME     | TARGET PORT |          URL          |
|-----------|---------------|-------------|-----------------------|
| default   | hola-minikube |             | http://10.0.2.15:31789 |
|-----------|---------------|-------------|-----------------------|
Opening service default/hola-minikubev2 in default browser...
open url failed: http://10.0.2.15:31789: exec: "xdg-open": executable
   file not found in $PATH
```

   Ignoramos el error y podemos acceder el servicio web en un navegador dentro de la máquina aasr1 de virtualbox. Si quisiésemos acceder al servicio desde fuera, tendríamos que redireccionar el puerto 31789 (o el que salga en cada caso, pues son aleatorios).

6. Para eliminar los contenedores y el clúster, tenemos los siguientes comandos:

```
$ kubectl delete services hola-minikube
$ kubectl delete deployment hola-minikube
$ minikube stop
$ minikube delete
```

7. Disponemos también de un interface web para gestionar *Minikube*:

```
$ minikube dashboard
```

## 6.6. Instalación de Xen

Por complitud incluimos aquí la configuración de Xen para la distribución Debian buster, si bien Xen no es un virtualizador de contenedores. Xen instala un hipervisor para controlar máquinas virtuales completas y realiza virtualización asistida por hardware si nuestro procesador tiene extensiones de virtualización o bien paravirtualización si no las tiene.

Las imágenes virtuales pueden estar contenidas en ficheros de imagen o en volúmenes lógicos. En este apartado utilizaremos el gestor de vólumenes lógicos LVM, que nos permite almacenar cada máquina virtual en su propio volumen lógico. Los pasos para instalar Xen son los siguientes:[16]

1. Primero debemos configurar nuestro sistema para que podamos conectar máquinas virtuales a la red externa. Esto se hace creando un puente en el `dom0` (el S.O. que ejecuta el hipervisor) que toma los paquetes de las máquinas virtuales y los reenvía a la red física para que puedan conectarse entre sí y con el exterior:

   ```
   $ apt-get install bridge-utils
   ```

   y modificando el fichero `/etc/network/interfaces` para que el interface Ethernet principal, que en nuestra máquina se denomina *ens3* (ethernet network slot 3) y el interface puente queden de la siguiente forma:

   ```
   # The primary network interface
   auto ens3
      iface ens3 inet manual

   auto xenbr0
   iface xenbr0 inet dhcp
       bridge_ports ens3
   ```

   Y reiniciar la red con: `systemctl restart networking`

2. A continuación instalamos los paquetes de Xen de la distribución:

   ```
   $ apt-get install xen-system-amd64  xen-utils xen-tools
   ```

   Reinciamos, eligiendo la tercera línea del menú de arranque, que indica *Debian GNU/Linux con el hipervisor Xen.*

[16] Adaptado de https://wiki.xenproject.org/wiki/Xen_Project_Beginners_Guide.

3. Una vez arrancado, comprobamos que Xen está funcionando:

| | |
|---|---|
| `xl info` | da información sobre el hipervisor (dom0) incluyendo la versión, memoria libre |
| `xl list` | enumera la ejecución de dominios, sus IDs, memoria, estado y tiempo de CPU consumidos |
| `xl top` | similar al programa `top` de Linux, muestra la ejecución de dominios en tiempo real como el uso de CPU, el uso de memoria y el acceso a dispositivos |

4. Para finalizar, utilizaremos las *xen-tools* para crear una máquina virtual de forma casi automática:

```
$ xen-create-image --hostname=stretch --memory=512mb --vcpus=1
   --lvm=nombre_apellido1_apellido2 --dhcp --pygrub --dist=stretch
```

donde los parámetros del dominio (máquina virtual) son autoexplicativos: hostname, RAM, número de CPUs, nombre del volumen lógico que se creará para almacenar la máquina virtual, inicio de la red por dhcp, gestor de arranque y distribución Debian.

**Importante:** Anotar la clave de root para poder usar la máquina.

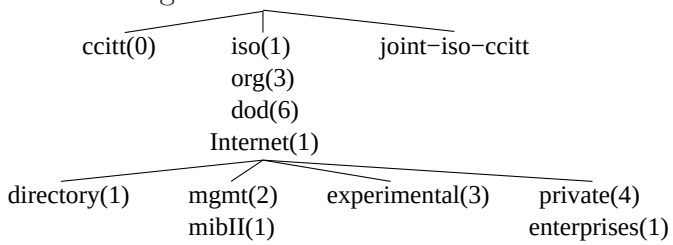
Y arrancamos el dominio con:

```
$ xl create -c /etc/xen/stretch.cfg
```

Desde otra consola, volver a ejecutar los comandos `xl info`, `xl list` y `xl top`. La máquina virtual la podemos apagar desde dentro con `poweroff`.

# 7. Gestión de redes con SNMP

SNMP es el protocolo estándar para la gestión de una red. SNMP trabaja con objetos cada uno de los cuales representa una información del sistema y que se agrupan en una MIB (Management Information Base). En esta MIB se puede leer o escribir de forma remota. La estructura de la MIB es jerárquica con objetos propuestos por diversas organizaciones:




Por ejemplo, los objetos estándar de Internet (MIB-II) tienen el prefijo .1.3.6.1.2.1. Estos se subdividen de la siguiente forma:

| | | | | |
|---|---|---|---|---|
| system (1) | interfaces (2) | at (3) | ip (4) | icmp (5) |
| tcp (6) | udp (7) | egp (8) | transmission (10) | snmp (11) |

Por ejemplo, para el objeto de OID (Object Id.) .1.3.6.1.2.1.4.3:

| | |
|---|---|
| Object Type: | **IpInReceives** |
| Object Identifier (OID): | .1.3.6.1.2.1.4.3 |
| Access: | read-only |
| Systax: | Counter32 |
| Description: | número total de datagramas recibidos |

## 7.1. El agente SNMP

Vamos a usar una simulación de Qemu con debian sarge. Antes de nada debemos tener la red funcionado (interfaces, rutas, DNS). Vamos a usar la aplicación *net-snmp*[17], que supondremos ya instalada. El fichero de configuración es *snmpd.conf* que puede localizarse en */etc/snmp*. Un ejemplo de configuración (ver *man snmpd.conf*) es:

```
syslocation  univ_santiago
syscontact  administrador_red
# rwuser user [noauth|auth|priv] [restriction_oid]
rwuser  debian auth .1.3.6.1
# rouser user [noauth|auth|priv] [restriction_oid]
rouser  pepito auth .1.3.6.1.2.1
# rwcommunity community [default|hostname|network/bits] [oid]
rwcommunity  comun_privada 127.0.0.1 .1.3.6.1
# rocommunity community [default|hostname|network/bits] [oid]
rocommunity  comun_publica 10.0.2.0/24 .1.3.6.1.2.1
```

[17] http://net-snmp.sourceforge.net y http://www.rederio.br/downloads/pdf/nt00601.pdf.

En este fichero se definen dos comunidades: *comun_publica* que puede leer el MIB desde cualquier ordenador de la red y *comun_privada* que puede modificarlos desde el ordenador local. El nombre de la comunidad se utilizará siempre en los comandos de consultas. (Para configuraciones más seguras consultar el manual).

El *agente SNMP* es el programa que recopila la información y la sirve a las aplicaciones que se la piden. Arrancamos el agente con */etc/init.d/snmpd restart*.

### 7.1.1. Comandos de consulta de la MIB

El comando básico es: `snmpwalk -v` *version ordenador* `-c` *comunidad OID*.

```
$ snmpwalk -v 2c localhost -c comun_privada .1.3.6.1.2.1.4.3
$ snmpwalk -v 2c 10.0.2.15 -c comun_publica .1.3.6.1.2.1.4.3
```

Algunas OID son (la base completa puede obtenerse omitiendo el OID):

| Grupo System (.1.3.6.1.2.1.1) | |
|---|---|
| sysDescr (.1.3.6.1.2.1.1.1) | Descripción del sistema |
| sysUpTime (.1.3.6.1.2.1.1.3) | Tiempo transcurrido desde arranque SNMP |
| sysContact (.1.3.6.1.2.1.1.4) | Persona de contacto |
| Grupo Interfaces (.1.3.6.1.2.1.2) | |
| ifNumber (.1.3.6.1.2.1.2.1) | Número de interfaces de red |
| ifDescr (.1.3.6.1.2.1.2.2.1.2) | Nombres de los interfaces |
| ifOperStatus (.1.3.6.1.2.1.2.2.1.8) | Estado de los interfaces |
| ifInOctets (.1.3.6.1.2.1.2.2.1.10) | Total de bytes recibidos por los interfaces |
| Grupo IP (.1.3.6.1.2.1.4) | |
| ipForwarding (.1.3.6.1.2.1.4.1) | Indica si se trata de un router |
| ipInReceives (.1.3.6.1.2.1.4.3) | Número total de datagramas recibidos |
| ipInHdrErrors (.1.3.6.1.2.1.4.4) | Datagramas recibidos con errores |
| Grupo ICMP (.1.3.6.1.2.1.5) | |
| icmpInMsgs (.1.3.6.1.2.1.5.1) | Número total de paquetes ICMP recibidos |
| icmpOutMsgs (.1.3.6.1.2.1.5.14) | Número total de paquetes ICMP enviados |
| Grupo TCP (.1.3.6.1.2.1.6) | |
| tcpMaxConn(.1.3.6.2.1.6.4) | Máximo de conexiones de TCP soportadas |
| tcpActiveOpens (.1.3.6.2.1.6.5) | Puertos TCP abiertos |
| tcpOutSegs (.1.3.6.2.1.6.11) | Número de segmentos enviados |
| Grupo UDP (.1.3.6.1.2.1.7) | |
| udpInDatagrams (.1.3.6.1.2.1.7.1) | Número total de datagramas UDP recibidos |
| udpNoPorts (.1.3.6.1.2.1.7.2) | Datagramas UDP a puerto incorrecto |
| udpLocalPort (.1.3.6.1.2.1.7.5.1.2) | Puertos UDP abiertos |
| Grupo SNMP (.1.3.6.1.2.1.11) | |
| snmpInPkts (.1.3.6.1.2.1.11.1) | Paquetes recibidos por el agente SNMP |
| snmpOutPkts (.1.3.6.1.2.1.11.2) | Paquetes enviados por el agente SNMP |
| snmpInTotalReqVars (.1.3.6.1.2.1.11.13) | Total de objetos proporcionados |

El OID se puede dar también en formato texto, por ejemplo:

`$ snmpwalk -v 2c` *localhost* `-c` *comun_privada* ipInReceives
`$ snmpwalk -v 2c` *localhost* `-c` *comun_privada* IP-MIB::ipInReceives

El siguiente comando también da la información del OID, pero este debe especificarse como el final la cadena. Por ejemplo, .1.3.6.1.2.1.2.2.1.2.1, ifDescr.1, IF-MIB::ifDescr.1 (terminado en .x). También podemos obtener el siguiente OID.

`$ snmpget -v` *version localhost* `-c` *comunidad OID*
`$ snmpgetnext -v` *version localhost* `-c` *comunidad OID*

### 7.1.2. Comando de traducción de OIDs

El comando de traducción de OIDs de formato númerico a formato texto es `snmptranslate` *OID*. Por defecto toma un OID en formato númerico, pero con la opción `-On` toma el OID puede darse en formato texto (aquí hay que incluir el prefijo SNMPv2-MIB::). Además, la opción `-Td` da información extendida, `-Of` muestra el OID sin abreviar y `-Tp` muestra el árbol a partir del OID indicado. Algunos ejemplos:

```
$ snmptranslate .1.3.6.1.2.1.1.1          # -> SNMPv2-MIB::sysDescr
$ snmptranslate -On SNMPv2-MIB::sysDescr  # -> .1.3.6.1.2.1.1.1
$ snmptranslate -Td .1.3.6.1.2.1.1.1
$ snmptranslate -Td -On SNMPv2-MIB::sysDescr
$ snmptranslate -Of .1.3.6.1.2.1.1.1
$ snmptranslate -Tp .1.3.6.1.2.1.1
```

Si no sabemos exactamente el nombre del OID, podemos usar las opciones `-IB` (mejor coincidencia) o `-TB` (todas las posibilidades):

```
$ snmptranslate -Ib 'sys.*ime'
$ snmptranslate -TB 'sys.*ime'
```

### 7.1.3. Comando de modificación de la MIB

El comando de escritura es `snmpset`:

`$ snmpset -v` *version localhost* `-c` *comunidad OID* `[i|u|t|a|o|s|x|d|n]` *dato*

El dato puede tener los siguientes tipos (ver la lista con `snmpset -h`):

```
i: INTEGER, u: unsigned INTEGER, t: TIMETICKS, a: IPADDRESS
o: OBJID, s: STRING, x: HEX STRING, d: DECIMAL STRING
U: unsigned int64, I: signed int64, F: float, D: double
```

Por ejemplo:

```
$ snmpget -v 2c -c comun_privada localhost sysName.0
$ snmpset -v 2c -c comun_privada localhost sysName.0 s "Sarge"
$ snmpget -v 2c -c comun_privada localhost sysName.0
```

## 7.2. Ejercicios no entregables:

1. Obtener desde snmp:
   (a) El número de interfaces de red (ifNumber) y sus nombres (ifDescr).
   (b) Dir. Ethernet (ifPhysAddress) y velocidades (ifSpeed) de los interfaces.
   (c) Las direcciones IP (ipAdEntAddr), máscaras (ipAdEntNetMask), direcciones de broadcast (ipAdEntBcastAddr) de los interfaces y las rutas (ipRouteDest, ipRouteIfIndex, ipRouteNextHop, ...).
   (d) Paquetes de ECO recibidos (icmpInEchos) y enviados (icmpOutEchos).
   (e) Los puertos TCP (tcpConnState) y UDP abiertos (udpLocalPort).
2. Obtener con *snmptranslate* la información extendida del OID de las direcciones Ethernet (ifPhysAddress) y de las velocidades (ifSpeed).
3. Obtener la lista de direcciones Ethernet y su velocidad con los comandos `snmpget` y `nmpgetnext`. Como OID hay que usar la cadena completa, bien numérica o bien de texto con el .x final.
4. Obtener algún OID de forma remota desde otro Qemu.
5. Leer el fichero de configuracion original (sección *Process checks*) e incorporar a snmp la capacidad de testear el número de procesos de cada tipo que están corriendo, por ejemplo *bash* y *getty*, poniendo un máximo y un mínimo. Reiniciar el servicio y probarlo con `snmpwalk` en el OID que nos indican.
6. Leer la seccción *Executables/scripts* y probar los ejemplos.
7. Leer la sección *disk checks* y obtener el % de uso de disco.
8. Lo mismo para la sección *load average checks*
9. Leer la sección *extensible section* y añadir a snmp la capacidad de ejecutar un comando, por ejemplo, `/bin/ps`, asignándole el OID .1.3.6.1.4.1.2000.50. Hacer lo mismo para un script escrito por el usuario, `/tmp/shtest`.

## 7.3. Interface gráfico

La navegación por los OID se puede realizar mediante el interface gráfico que viene junto al paquete net-snmp. La aplicación se llama con `tkmib` y los parámetros a configurar son la comunidad (en el menú opciones de snmp 1/2c) y el ordenador a monitorizar (pequeño rectángulo junto a los botones). En la ventana superior se selecciona el OID desplegando el árbol, en las ventanas intermedias se muestra la descripción del OID y con el botón *getnext* obtenemos la información. Con *walk* y *table* podemos obtener información cuando hay varios elementos (por ejemplo, las direcciones Ethernet de varios interfaces).

# 8. Monitorización de redes con MRTG

La gestión de redes con SNMP sería muy tediosa si hubiera que hacerla en modo texto con comandos. Afortunadamente existen aplicaciones como MRTG (Multi Router Trafic Graph)[18] que nos permiten representar de forma gráfica en una página web la información que nos da snmp. Aunque inicialmente MRTG fue creado para representar el tráfico que atravesaba los interfaces de los routers, podemos visualizar prácticamente cualquier dato.

## 8.1. Configuración

La configuración de `mrtg` se realiza con una utilidad del propio programa, `cfgmaker`. Para crear la página html básica en el directorio */var/www/mrtg* ejecutamos otra utilidad que se llama `indexmaker`:

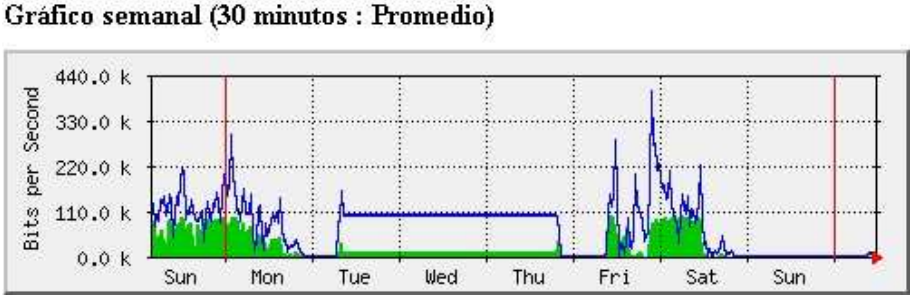
```
$ cfgmaker --community comun_privada --output /etc/mrtg.cfg localhost
  # editar /etc/mrtg.cfg para definir el "WorkDir" como /var/www/mrtg
$ indexmaker --output /var/www/mrtg/index.html /etc/mrtg.cfg
```

A continuación, lanzamos el programa (ignorar el Rateup WARNING:) e iniciamos el servidor web *apache*:

```
$ mrtg /etc/mrtg.cfg
$ /etc/init.d/apache restart
```

Desde un navegador del PC del laboratorio cargamos la página de dirección http://localhost:8000/mrtg/. Pinchando en el gráfico vamos a una subpágina que muestra gráficas diarias, semanales, etc. Si todo funciona, podemos lanzar el proceso del sistema y la gráfica se irá construyendo con el paso del tiempo.

```
$ mrtg /etc/mrtg.cfg --daemon
```




Por defecto, MRTP muestra el tráfico en cada uno de los interfaces de red, pero podemos emplearlo para monitorizar cualquier OID (numérico). Por ejemplo, para monitorizar el número de conexiones TCP abiertas, añadimos al final del fichero */etc/mrtg.cfg* lo siguiente:

[18] http://libertonia.escomposlinux.org/story/2003/1/17/224253/241.

```
Target[conexiones]: .1.3.6.1.2.1.6.9.0&.1.3.6.1.2.1.6.9.0:comun_privada@localhost
Options[conexiones]: gauge
Title[conexiones]: Conexiones TCP abiertas
PageTop[conexiones]: <h1>Conexiones TCP abiertas</h1>
MaxBytes[conexiones]: 1000000
Background[conexiones]: #FF0000
YLegend[conexiones]: # conex.
ShortLegend[conexiones]:
LegendI[conexiones]: Conexiones:
LegendO[conexiones]:
Legend1[conexiones]: Conexiones TCP abiertas
```

Nótese que al usar la misma variable dos veces, se va a pintar únicamente un gráfico. Lo normal es tener dos variables distintas y pintar dos gráficos en la misma imagen. La opción *gauge* le dice a mrtg que su valor es un contador absoluto, o sea que su valor es siempre el que tiene que representar (por ejemplo, la cantidad de memoria usada). En caso contrario, mrtg cree que es un contador normal, con lo cual resta el valor actual del valor anterior (por ejemplo, el número de paquetes transmitidos en un intervalo).

A continuación, ejecutamos de nuevo el comando `indexmaker` con las mismas opciones que antes. No hay que ejecutar `cfgmaker` porque sobreescribiría nuestros cambios. Por último, reiniciamos `mrtg` (paramos y reiniciamos el proceso) o esperamos que se actualicen las páginas. Si ahora recargamos la página, debería aparecernos una nueva gráfica.

## 8.2. Sintaxis

La sintaxis de */etc/mrtg.cfg* se puede consultar en el manual con `man 1 mrtg-reference`[19]. Como opciones generales tenemos las siguientes:

`Refresh` indica cada cuantos sg el navegador recargará la página.
`Interval` especifica cada cuantos minutos mrtg tomará datos.

Cada uno de los gráficos se identifica por un *id* y tiene los siguientes parámetros obligatorios:

`Target[id]`: con el formato OID1&OID2:comunidad@host, donde las dos OID a mostrar pueden darse en formato numérico o texto, pero deben ser el final de la cadena; se puede comprobar que son correctas con el comando `snmpget`.
`MaxBytes[id]`: El tamaño máximo de almacenamiento para datos recopilados.
`Title[id]`: Título de la página web que contendrá los gráficos.

Los siguientes parámetros son opcionales:

[19] Ver los ejemplos en la referencia anterior y en http://www.adslayuda.com/3ComW-11.html, http://net-snmp.sourceforge.net/tutorial/tutorial-5/mrtg/index.html y http://howto.aphroland.org/HOWTO/MRTG/SystemMonitoring.

`PageTop[id]:` y `PageFoot[id]:` Texto a escribir en la página web.
`XSize[id]:` e `YSize[id]::` Tamaño de los gráficos, Xsize debe estar comprendido entre 20 y 600 e Ysize debe ser mayor que 20. Por defecto los gráficos son de 400 por 100 píxeles.
`kilo[id]:` Para los multiplicadores de la unidades (k, M y G); su valor por defecto es 1000 pero se puede cambiar a 1024.
`Colours[id]:` Cambia los colores por defecto y deben especificarse 4 en el formato Col1#RRGGBB,Col2#RRGGBB,Col3#RRGGBB,Col4#RRGGBB.
`Background[id]:` Color de fondo en el formato #RRGGBB.
`YLegend[id]:` Etiqueta para el eje Y.
`Legend[1234IO][id]:` Etiquetas para los distintos colores en la subpágina.
`ShortLegend[id]:` Etiqueta (por defecto "b/s") para las unidades de Max, Average y Current en la subpágina.

`Options[id]`: Las posibilidades son *growright* (para que el gráfico crezca a la derecha; por defecto crece hacia la izquierda), bits (muestra cantidades en bits, pues por defecto los muestra en bytes; equivale a ×8), *perminute* (valores por minuto en vez de por sg; equivale a ×60), *perhour* (×360), *gauge* (muestra valores absolutos y no diferencias con el valor anterior, que es el defecto).

⇒ **ENTREGA 7**. Añadir una segunda gráfica a la página web con el tiempo de CPU usado (ssCpuRawUser) y libre (ssCpuRawIdle), sin borrar las antiguas. Obtener los OID, en los cuales hay que añadir un último dígito que indica el número del procesador, y comprobar que funciona con el comando *snmpget*, puesto que este comando es el que llamará MRTG. El código deberá tener un *id* diferente al de las gráficas anteriores.

## 8.3. Ejercicios no entregables:

1. Mostrar la gráfica de la memoria libre en el ordenador. Usar los comandos snmp para obtener el OID partiendo de .1.3.6.1.4.1.2021

2. Mostrar la gráfica del espacio ocupado en la partición raíz del disco. Usar los comandos snmp para obtener el OID y su significado partiendo de hrStorageUsed.

3. Iniciar un segundo Qemu, ejecutar en él SNMP y desde el primer Qemu mostrar la gráfica de algún OID del segundo Qemu. Tal como lo tenemos configurado, tenemos que usar la comunidad pública.