Arquitecturas Virtuales

Práctica 6.- Prestaciones de la virtualización

Eladio Gutiérrez Carrasco – Julián Ramos Cózar
Diciembre , 2021

Introducción

El objetivo de esta práctica es medir experimentalmente la eficacia de la virtualización y comprobar el impacto de diferentes técnicas sobre el rendimiento.

Medios Materiales

Esta práctica se realizará sobre el sistema operativo Linux instalado en los equipos del laboratorio que dispone del software de virtualización VirtualBox de Oracle[1].

VirtualBox es un virtualizador tipo 2 (*hosted hypervisor*).

Como material de partida se suministrará una máquina virtual Linux, con la distribución Lubuntu 12.04[2] de 32 bits instalada y configurada.

El usuario de trabajo y su contraseña en esta máquina virtual es el siguiente:

```
username: ubuntu
password: ubuntu
```

Para facilitar su uso, tras el arranque de la máquina, se efectúa un *autologin* de sesión en una consola de texto con este usuario. La configuración de `sudo` (`/etc/sudoers`) permite ejecutar comandos como administrador (root) a este usuario sin necesidad de contraseña.

El entorno gráfico se puede iniciar arrancando el siguiente servicio:

```
$ sudo service lightdm start
```

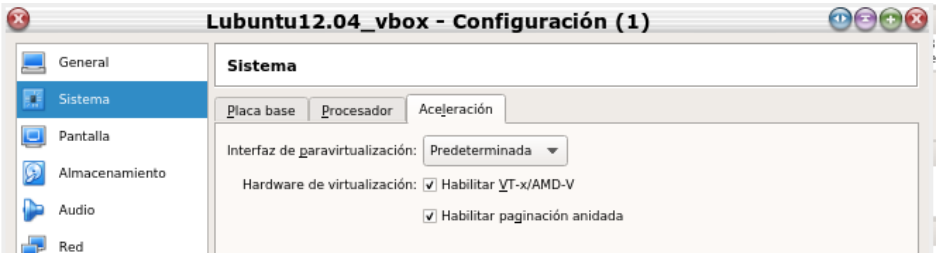
Para obtener la versión del kernel y del S.O. tanto del *host* como del *guest*, teclear desde el *shell*:

```
$ uname –a
$ cat /etc/*release
```

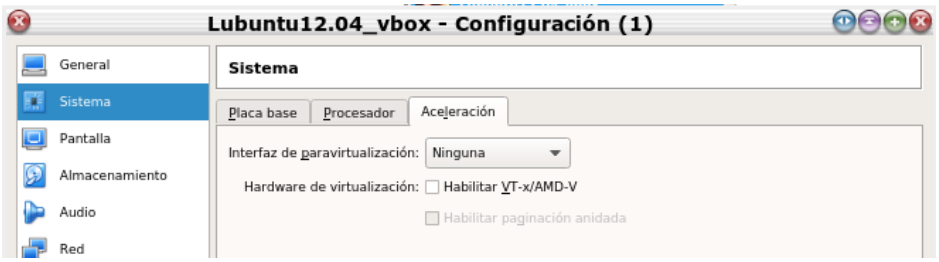
[1] Se ha elegido este software de virtualización porque aún en sus últimas versiones permite ejecutar máquinas *guest* de 32 bits sin *habilitar* el soporte hardware virtualización de la CPU. Otros productos como VMware han abandonado la traducción binaria y sólo permiten ejecutar máquinas virtuales con soporte hardware.
[2] Lubuntu es una versión de Ubuntu con el interfaz ligero de escritorio LXDE

En esta práctica se propone analizar el rendimiento de los dos métodos básicos de virtualización. La selección de dicho método se realiza en la configuración de la CPU de la máquina virtual:

- *Trap and Emulate con soporte HW*: Para ello en la pestaña de configuración "Sistema" donde se configura la CPU de la máquina, activaremos las opciones de "Hardware de virtualización" (VT-x, paginación anidada). Se recomienda poner el "Interfaz de paravirtualización" a "Predeterminada" (por defecto).



- *Binary Translation*: Para ello desactivamos las opciones de aceleración que activamos para la configuración anterior.



Metodología

Nuestro objetivo es obtener figuras de mérito de la ejecución de aplicaciones con diferentes características, en las máquinas *guest* configuradas con las técnicas de virtualización que nos ofrece VirtualBox: con aceleración (soporte VT-x) y sin aceleración (traducción binaria dinámica).

Recordemos que la arquitectura IA32 original de Intel no es virtualizable en sentido estricto y por ello era necesario un módulo de traducción binaria para 32 bits. Las modernas CPUs x86-64 pueden ejecutar código de 32 bits ya que mantienen compatibilidad binaria, y el soporte hardware de virtualización puede activarse o no.

La eficacia de ambas técnicas las compararemos, así mismo, con la ejecución en la máquina *host*. Se trata de analizar los diferentes factores que se ponen en juego.

En particular nos vamos a centrar en dos métricas: el tiempo de ejecución (que es una medida absoluta) y el número de instrucciones ejecutadas, que nos da una idea del *overhead* introducido por el hipervisor.

Hemos de tener en cuenta que VirtualBox es un *hosted hypervisor*, implementación típica de los virtualizadores de escritorio. Por ello, para cada máquina virtual existe un proceso que corresponde con el hipervisor. Es fácil localizar dicho proceso, e identificarlos en el host, a partir del nombre del hipervisor (VirtualBox). He aquí un ejemplo de una instancia en ejecución:

```
user@host$ ps -edfl | grep -i virtualbox
4 S user  22329 [...] /usr/lib/virtualbox/VirtualBox --comment Lubuntu12.04_vbox --startvm [...]
```

En este ejemplo el hipervisor de la máquina *guest* en ejecución tiene como pid, 22329. Observa que como argumento del hipervisor, aparece el nombre del fichero de configuración de la máquina en cuestión (Lubuntu12.04_vbox, en este ejemplo).

Dado que no está garantizado una medida exacta del tiempo en la máquina *guest*, un aspecto importante en nuestra metodología, es intentar realizar todas las mediciones en el *host*, por lo que vamos a necesitar un mecanismo de lanzar comandos en las máquinas virtuales desde el host.

En esta práctica hemos optado por la aplicación *Secure Shell*, **ssh**, dada su disponibilidad, popularidad, versatilidad y que entre otras cosas nos proporciona la salida estándar de los comandos ejecutados remotamente.

> Nota: tanto las máquinas virtuales proporcionada, como el *host* del laboratorio tienen ya instalado el cliente y servidor `ssh`.
>
> No obstante, forma parte de la paquetería estándar de la mayoría de distribuciones Linux. En plataformas ubuntu se puede instalar fácilmente con:
>
> ```
> apt-get install ssh
> ```

La sintaxis para ejecutar un comando en una máquina remota mediante ssh[3]:

```
ssh user@[remote host ip] <command> # Este commando se ejecutará en la máquina remota
```

Sin más configuración, ssh nos solicitará la *password* del usuario remoto cada vez, por lo que es conveniente de cara a los experimentos que vamos a realizar, configurar el acceso por medio de una autenticación por clave pública/privada. En el **anexo I**, se explica cómo hacerlo.

En este punto encenderemos la máquina virtual de nuestro experimento, bien con aceleración (*trap & emulate*) o sin ella (*BT*), y recopilaremos su dirección ip[4], que será necesaria para ejecutar comandos remotos. También obtendremos los *pid* de las instancias del hipervisor VirtualBox, que serán necesarios para realizar las mediciones. Ten cuidado, ya que aunque la ip se mantiene normalmente entre arranques de la máquina virtual (es política del DHCP del nic virtual), el identificador de proceso cambia cada vez. Comprobaremos también que se permite la ejecución de comandos remotos con ssh.

Máquina virtual	IP	PID hipervisor (proceso VirtualBox)
Experimento con aceleración (VT-x, *Trap and Emulate)*		
Experimento sin aceleración (*Binary Translation*)		

Una ver que hemos elegido un método para lanzar las pruebas en las máquinas virtuales, tenemos disponibles diferentes opciones para realizar las mediciones de rendimiento.

- Comando interno **time**: proporciona un resumen de los tiempos de ejecución.

```
$ time ls
forkwait  forkwait.c  ioport  ioport.c  myrt  myrt.c
real 0m0.003s
user 0m0.000s
sys  0m0.000s
```

[3] El protocolo ssh emplea el puerto 22 de tcp, para establecer la comunicación con el sistema remoto. Si existiera algún firewall en las máquinas virtuales, o en el host, debe tenerse en cuenta que hay que permitir dicho puerto.

[4] Para obtener la configuración ip de una máquina linux usar el comando: `sudo ifconfig` o bien `ip address`

- Comando externo **/usr/bin/time**: proporciona un resumen más detallado de la contabilidad de eventos originados durante la ejecución del proceso

```
$ /usr/bin/time -v ls
forkwait  forkwait.c  ioport  ioport.c       myrt  myrt.c
    Command being timed: "ls"
    User time (seconds): 0.00
    System time (seconds): 0.00
    Percent of CPU this job got: 0%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
    Average shared text size (kbytes): 0
    Average unshared data size (kbytes): 0
    Average stack size (kbytes): 0
    Average total size (kbytes): 0
    Maximum resident set size (kbytes): 3840
    Average resident set size (kbytes): 0
    Major (requiring I/O) page faults: 0
    Minor (reclaiming a frame) page faults: 308
    Voluntary context switches: 1
    Involuntary context switches: 6
    Swaps: 0
    File system inputs: 0
    File system outputs: 8
    Socket messages sent: 0
    Socket messages received: 0
    Signals delivered: 0
    Page size (bytes): 4096
    Exit status: 0
```

- Monitor de rendimiento ***perf***:

  Este monitor de rendimiento para linux, permite realizar análisis de rendimiento (*profiling*) bastante detallados haciendo uso de los contadores de eventos hardware disponibles en el procesador.

  En distribuciones como ubuntu, el comando *perf* estará disponible tras la instalación del paquete *linux-tools*.

  En este URL https://perf.wiki.kernel.org/index.php/Tutorial#Counting_with_perf_stat podemos consultar un tutorial de las diferentes opciones y usos de esta herramienta. Así mismo, en línea de comandos podemos obtener información del comando con `perf --help` ó `perf stat --help`.

  En nuestros experimentos, estamos interesados en la opción de estadísticas resumidas de los contadores.
  El siguiente ejemplo aclara su uso:

```
$ perf stat ls
forkwait  forkwait.c  ioport  ioport.c   myrt  myrt.c

 Performance counter stats for 'ls':

          1,153774 task-clock                #    0,765 CPUs utilized
                 4 context-switches          #    0,003 M/sec
                 1 CPU-migrations            #    0,001 M/sec
               273 page-faults               #    0,237 M/sec
         1.826.908 cycles                    #    1,583 GHz
         1.388.371 instructions              #    0,76  insns per cycle
           285.446 branches                  #  247,402 M/sec
            12.720 branch-misses             #    4,46% of all branches

       0,001508994 seconds time elapsed
```

Una opción interesante de la herramienta es monitorizar un proceso en ejecución, conocido su pid. Así, podemos monitorizar el proceso hipervisor VritualBox de una máquina virtual en ejecución, como indica el ejemplo:

```
$ sudo perf stat --pid 22329
 Performance counter stats for process id '22329':

          2,626681 task-clock                #    0,002 CPUs utilized
               129 context-switches          #    0,049 M/sec
                 0 CPU-migrations            #    0,000 M/sec
                 0 page-faults               #    0,000 M/sec
         3.742.567 cycles                    #    1,425 GHz
         1.205.609 instructions              #    0,32  insns per cycle
           239.920 branches                  #   91,340 M/sec
            43.452 branch-misses             #   18,11% of all branches

       1,168088329 seconds time elapsed
```

Observa que el pid corresponde con uno de los procesos *VirtualBox*, identificado con anterioridad.

Con la opción **--pid**, la monitorización termina con la señal de interrupción **SIGINT**, que enviamos mediante la pulsación de **^C** en el terminal. Las estadísticas mostradas corresponden a las que se han recolectados desde que se lanzó el comando *perf*, hasta la recepción de SIGINT.

De estas tres opciones nos quedaremos con la herramienta *perf,* ya que nos permite monitorizar el hipervisor y obtener estadísticas como el número de instrucciones, además del tiempo.

¿Cuál será nuestra metodología para medir las estadísticas de rendimiento de un proceso ejecutado en el hipervisor?

Seguiremos una estrategia en tres pasos, desde el sistema operativo *host*:

1. Lanzamos el *profiling* del hipervisor, dejándolo en *background*[5]:

```
$ sudo perf stat --log-fd 2 --pid [PID proceso VirtualBox] &
```

Nota: podemos seleccionar los eventos cuyas estadísticas queremos mostrar. Una selección de interés podría ser la siguiente:

```
$ sudo perf stat -e context-switches,cpu-migrations,page-faults,task-
clock,cycles,instructions:u,instructions:k,instructions --pid [PID proceso
VirtualBox] &
```

2. Ejecutamos remotamente la aplicación en la máquina virtual en cuestión:

```
$ ssh user@[ip máquina virtual] [comando]
```

[5] La opcion "`--log-fd 2`" es necesaria en algunas instalaciones, para evitar el error "*Failed opening logfd: Invalid argument*".

3. Detenemos el *profiling* para volcar las estadísticas (el comando **pkill** permite enviar una señal a todos los procesos cuyo nombre de ejecutable coincida con una expresión regular):

```
$ sudo pkill –INT perf
```

(¡ojo! ¡con esto último estamos aniquilando todos los procesos que contengan la cadena "*perf*" !)

Por ejemplo, veamos cuántas instrucciones requiere y el tiempo invertido por un "ls" en una máquina virtual, cuya dirección ip es 192.168.175.146 y su hipervisor VirtualBox asociado es el proceso 22329 en el sistema operativo host:

```
$ sudo perf stat --pid 22329 & ssh user@192.168.175.146 "ls" ; sudo pkill -INT perf
[1] 27843
Desktop forkwait forkwait.c ioport ioport.c pi.bc

 Performance counter stats for process id '22329':

         73,662614 task-clock                #    0,019 CPUs utilized
             1.853 context-switches          #    0,025 M/sec
                92 CPU-migrations            #    0,001 M/sec
                54 page-faults               #    0,001 M/sec
       259.524.876 cycles                    #    3,523 GHz
        75.259.481 instructions              #    0,29  insns per cycle
        15.501.382 branches                  #  210,438 M/sec
           974.984 branch-misses             #    6,29% of all branches

              3,85 seconds time elapsed
```

Observa el uso de los separadores "&" y ";" en la línea de comandos del shell. Sustituye la ip y el pid del hipervisor de la máquina virtual por sus correspondientes valores.

Algunas cosas a tener en cuenta y que afectan a la precisión de las medidas:

- Existe un efecto de *warm up* la primera vez que se lanza la aplicación (carga de librerías compartidas, caché/buffers de ficheros, ...), por lo que la primera ejecución puede no ser significativa.
- Hay un *overhead* adicional en la medida debido a la propia ejecución a través de ssh, ya que en la máquina remota el demonio *sshd* debe atender la petición y abrir un shell en el que se lanza la aplicación bajo test.
- Hay un "ruido de fondo" debido a que existe más actividad en el hipervisor aparte de ejecutar la aplicación de interés, como por ejemplo atender las interrupciones, paquetes que llegan de la red, servicios en segundo plano, otros procesos, etc.
- Podemos estimar una parte de todo este *overhead* repitiendo el proceso de medida anterior para el comando `exit` (es decir la prueba nula), o con más precisión mediante una ejecución de `sleep` con la duración de la prueba.
- Debe comprobarse que la carga de la máquina remota es cercana a cero para que la medida sea significativa. Recuerda que se puede testear la carga con comandos como:

```
uptime
top
htop
```

Para facilitar la realización de los experimentos, en un shell del host guardaremos los datos de interés en variables y definiremos una función para lanzar las pruebas:

```
#Máquina virtual 1, configurada con aceleracion (VT-x)
#Máquina virtual 2, configurada sin aceleración (BT)
export vmip1=[dirección ip máquina virtual 1]
export vmpid1=[pid del hipervisor de la máquina 1]

export vmip2=[dirección ip máquina virtual 2]
export vmpid2=[pid del hipervisor de la máquina 2]

#Salida por defecto
vmperfd() { sudo perf stat --log-fd 2 --pid $1 &
           ssh user@$2 $3;
           sudo pkill -INT perf; }

#Una selección de eventos
vmperf() { eventos='context-switches,cpu-migrations,page-faults,task-
clock,cycles,instructions:u,instructions:k,instructions';
           sudo perf stat -e $eventos --pid $1 &
           ssh user@$2 $3;
           sudo pkill -INT perf; }
```

Con lo cual el experimento del ejemplo anterior resultará ahora tan fácil de teclear como:

```
vmperf $pid1 $ip1 "ls"
```

Microbenchmarks

Utilizaremos tres microbenchmarks con comportamientos diferenciados, en cuanto a su comportamiento y exigencia de recursos.

- Microbenchmark 1 (*CPU bound*): cálculo del número π con la calculadora de precisión arbitraria **bc**

```
#!/usr/bin/bc -l
scale=2000
pi=4*a(1)
pi
quit
```

- Microbenchmark 2 (*Syscall bound*): ejecución intensiva de llamadas al sistema

```
#include <stdio.h>
#define Nfork 50000
int main(int argc, char *argv[]) {
        int i, pid;
       for (i = 0; i < Nfork; i++) {
               pid = fork();
               if (pid < 0) {
                       printf("Fork #%i failed\n", i);
                       return -1;
                 }
               if (pid == 0) return 0;
               waitpid(pid, NULL, 0);
       }
       return 0;
}
```

- Microbenchmark 3 (*IO bound*): programa intensivo en operaciones de E/S a disco

```
#!/bin/sh
SIZE=100 #Tamaño en MB del fichero
dd if=/dev/zero of=/tmp/file.dat bs=1M count=$SIZE oflag=direct
```

Experimentos

Una expresión usada para evaluar la pérdida de eficiencia por virtualización, o *slowdown*, es la siguiente:

$$Slowdown = \frac{t_{virtual\ (guest)}}{t_{real\ (host)}} = f_p\ N_e + (1 - f_p)$$

donde, $t_{virtual}$ es el tiempo requerido por el programa en la máquina virtual, t_{real} el requerido en la máquina real, f_p es la fracción de instrucciones privilegiadas, *Ne* el número medio de instrucciones reales que reemplazan una instrucción virtual (bien por emulación o vía *traps*).

En estos experimentos ejecutaremos los 3 microbenchmarks, determinando en cada caso, con la mayor precisión posible, el tiempo de ejecución de cada uno y el número de ejecuciones que ha requerido el hipervisor para ello. Seguiremos la metodología descrita en el apartado anterior.

Completa la siguiente tabla, con los resultados observados. Para una mayor precisión, repite los experimentos varias veces, y promedia los resultados. Puedes ser más preciso determinando la desviación típica de las medidas. Descarta los datos de la primera ejecución de cada prueba.

Microbenchmark	Máquina HOST		Máquina virtual 1 *Trap & Emulate*		Máquina virtual 2 *Binary Translation*	
	elapsed time	# instr.	elapsed time	# instr.	elapsed time	# instr.
bc π						
forkwait						
ddcopy						

Tabla 1

Podemos modelar el *overhead* de una prueba, como el trabajo extra realizado lanzan el comando a través de ssh, más el "ruido de fondo" de ejecución del hipervisor durante el intervalo de tiempo que dura la monitorización:

Time overhead = τ *,*

donde τ representa el tiempo extra introducido por el lanzamiento del comando bajo test (principalmente debido a la ejecución remota con ssh),

Instruction overhead = $\alpha + \beta\ t$*,*

donde t es la duración de la prueba, α representa el número de instrucciones extra introducidas por el lanzamiento del comando y β representa el número de instrucciones por unidad de tiempo que el hipervisor ejecuta en reposo ("ruido de fondo").

Estima los parámetros de overhead de las ejecuciones[6], mediante diferentes ejecuciones del comando `sleep t`, con diferentes valores de t:

	HOST		Máquina virtual 1 *Trap & Emulate*		Máquina virtual 2 *Binary Translation*	
	elapsed time	# instr.	extra elapsed time (diferencia con host)	# instr. extra (diferencia con host)	extra elapsed time (diferencia con host)	# instr. extra (diferencia con host)
sleep 1						
sleep 5						
sleep 10						

Tabla 2

Anota la estimación de los parámetros de *overhead*, para cada máquina virtual

Máquina virtual 1 (Trap & Emulate)	**τ =**	**α =**	**β =**
Máquina virtual 2 (Binary Translation)	**τ =**	**α =**	**β =**

Introduce la corrección de los parámetros de *overhead* estimados en las medidas de las máquinas virtuales de la Tabla 1:

$t' = t - \tau$

$\#instr.' = \#inst - (\alpha + \beta\, t)$

Microbenchmark	Máquina virtual 1 *Trap & Emulate*		Máquina virtual 2 *Binary Translation*	
	elapsed time (SIN overhead)	# instr. (SIN overhead)	elapsed time (SIN overhead)	# instr. (SIN overhead)
bc π				
forkwait				
ddcopy				

Tabla 3

A partir de los datos determina ahora los parámetros de la ecuación:

Benchmark	Slowdown	f_p	*Ne*
bc π			
forkwait			
ddcopy			

Interpretación de los resultados y Observaciones:

Explica cualquier suposición, hipótesis o metodología que hayas considerado aparte de lo comentado en el guión.

Interpreta los resultados y anota cualquier particularidad que observes en los mismos, resultados paradójicos, explicaciones, etc.

[6] Para realizar un ajuste lineal puedes utilizar WolframAlfpha (http://www.wolframalpha.com/), introduciendo `linear fit {{x1,y1}, {x2,y2}, {x3,y3}, ...}}.`

Anexo I: Configuración de ssh

En el lado del cliente (*host*), crearemos un par clave pública / clave privada, con el objeto de poder hacer conexiones hacia el sistema remoto (MV *guest*) sin necesidad de introducir la *password* del usuario.

1. Crear las claves (DEJAR VACÍA LA PASSPHRASE)

```
$ ssh-keygen
     Generating public/private rsa key pair.
     Enter file in which to save the key (/home/practicas/.ssh/id_rsa):
     /home/practicas/.ssh/id_rsa already exists.
     Overwrite (y/n)? y
     Enter passphrase (empty for no passphrase):
     Enter same passphrase again:
     Your identification has been saved in /home/practicas/.ssh/id_rsa.
     Your public key has been saved in /home/practicas/.ssh/id_rsa.pub.
     The key fingerprint is:
     64:e6:28:05:fd:b8:58:cb:b2:9c:26:52:4c:de:85:51 practicas@lac-pr9
     The key's randomart image is:
     +--[ RSA 2048]----+
     |    .oE          |
     |    ...          |
     |     o.o+        |
     |  . ..+*.        |
     | + ..=.oS        |
     |  + +.+          |
     | . . +           |
     |. . =            |
     | . o             |
     +-----------------+
```

2. Propagar la clave pública hacia el sistema remoto (*guest*) (la misma clave pública se puede usar en todos los servidores ssh (*guest*) que se desee):

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@$vmip1
```

3. Si la propagación ha sido correcta, ahora podemos ejecutar comandos directamente, por ejemplo:

```
$ssh user@[remote host ip] uname -a
```

Para disminuir retrasos innecesarios en la conexión sigue este par de recomendaciones de configuración del servicio ssh en el lado del servidor (*guest*).

1. Edita el fichero de configuración de servidor ssh (en el sistema remoto)
   `/etc/ssh/sshd_config`

2. Configura la siguiente opción:
   `GSSAPIAuthentication no`

3. Configura la siguiente línea (importante):
   `UseDNS no`

4. Una vez modificado el fichero sshd_config, rearranca el demonio del servidor ssh:
   `sudo service ssh restart`

Otra recomendación que en ocasiones acelera la conexión ssh, es deshabilitar la carga módulos opcionales (para ello basta con comentar las líneas correspondientes con #) en el fichero de configuración de autenticación `/etc/pam.d/sshd`, por ejemplo algo como:

```
# session    optional    pam_motd.so  # deshabilitado
# session    optional    pam_mail.so
```

Anexo II: Otros aspectos a tener en cuenta y que pueden afectar a las ejecuciones

Existen múltiples factores que han de considerarse a la hora de abordar de forma exacta la medida de tiempos de ejecución de los programas. En ocasiones estos factores pueden alterar las medidas y proporcionar resultados muy diferentes a los esperados. Aquí se comentan algunos.

- Ejecución en multiprocesadores

  La ejecución en multiprocesadores, puede producir una migración de los *threads* del proceso entre los diferentes cores. Esto puede afectar al rendimiento del microbenchmark 2, puesto que el proceso padre puede moverse a otro core tras la llamada a *fork()*. Observa que la herramienta *perf* da una medida del número de migraciones sufridas por el proceso.

  La información de los procesadores disponibles en la máquina (cores) se puede consultar así:

  ```
  less /proc/cpuinfo
  ```

  Si deseamos establecer una afinidad de procesador y atar (*pin*) un proceso a una CPU dada podemos emplear el comando `taskset`.

  ```
  taskset -c 1 a.out #Lanza el programa a.out en el core número 1
  ```

- Sincronismo del reloj

  Las derivas del reloj de una máquina virtual, pueden hacer imprecisas las medidas realizadas en ellas. Esta no ha sido nuestra metodología porque todas las medidas se han realizado en el *host*.

  Hay dos formas principales de mantener el sincronismo:

  - Sincronización con el host; dicha sincronización la realiza las *tools* del hipervisor.
  - Sincronización con un servidor horario NTP.

  Podemos comprobar el sincronismo del reloj contra un servidor horario:

  ```
  ntpdate -q cronos.uma.es #cronos.uma.es es el servidor NTP de la UMA.
  ```

- Frecuencia de la CPU

  La frecuencia asociadas a los cores puede modificarse en tiempo de ejecución como consecuencia de políticas de ahorro de energía. A veces, en linux, los procesos en tiempo real pueden verse afectados por una reducción en la frecuencia.

Desde el shell de linux podemos monitorizar la frecuencia de un core en particular[7]:

```
#Monitorizar core 7
watch --interval 1 sudo  cat /sys/devices/system/cpu/cpu7/cpufreq/cpuinfo_cur_freq

#Monitorizar todos los cores
watch --interval 1 sudo  cat /sys/devices/system/cpu/cpu*/cpufreq/cpuinfo_cur_freq
```

En la distribución de linux disponible en los laboratorios (ubuntu) podemos activar/desactivar el ajuste automático de la frecuencia de cada procesador escribiendo la cadena "ondemand" o "performance" respectivamente en el siguiente parámetro de la configuración del kernel */sys/devices/system/cpu/cpu0/cpufreq/scaling_governor*

```
for i in {0..3}; do  #Suponiendo 4 cores
 #Esta línea pondría a los cores a su máxima frecuencia
 echo performance | sudo tee /sys/devices/system/cpu/cpu${i}/cpufreq/scaling_governor
done
```

- Otras consideraciones

  Hay que tener en cuenta que, para realizar una comparativa justa, los programas que se ejecutan han de ser lo más idénticos posibles en el *host* y en las máquinas *guest*. La comparativa puede verse afectada por:

  - que el *host* y el *guest* estén corriendo diferentes versiones de kernel, o que los parámetros de ejecución de éste no coincidan,
  - que el sistema *host* sea de 64 bits y el de la máquina virtual de 32,
  - que los programas estén compilados con distintas versiones de librerías (puede usarse `ldd` para comprobar las librerías a las que invoca un ejecutable),
  - que se hayan empleado opciones de compilación (optimizaciones, ...) diferente, cuando se compilaron los benchmarks en las diferentes máquinas,
  - que la frecuencia de los procesadores varíe entre las ejecuciones.

Anexo III: Configuración del interfaz de red en las máquinas virtuales

Se recomienda utilizar en los experimentos el interfaz de red "solo anfitrión" *(host-only)*, ya que permite una comunicación por red más eficiente entre el host y la máquina virtual.

Para ello en VirtualBox hay que:

- crear una nueva subred anfitrión (si no se tiene ninguna) en el gestor de máquinas virtuales,
- y posteriormente configurar la red de la máquina virtual añadiendo el adaptador "solo anfitrión"

En las siguientes capturas se ilustran estas acciones.

[7] La localización de esta información puede variar de una distribución de linux a otra, dentro del directorio /sys/ ó /proc

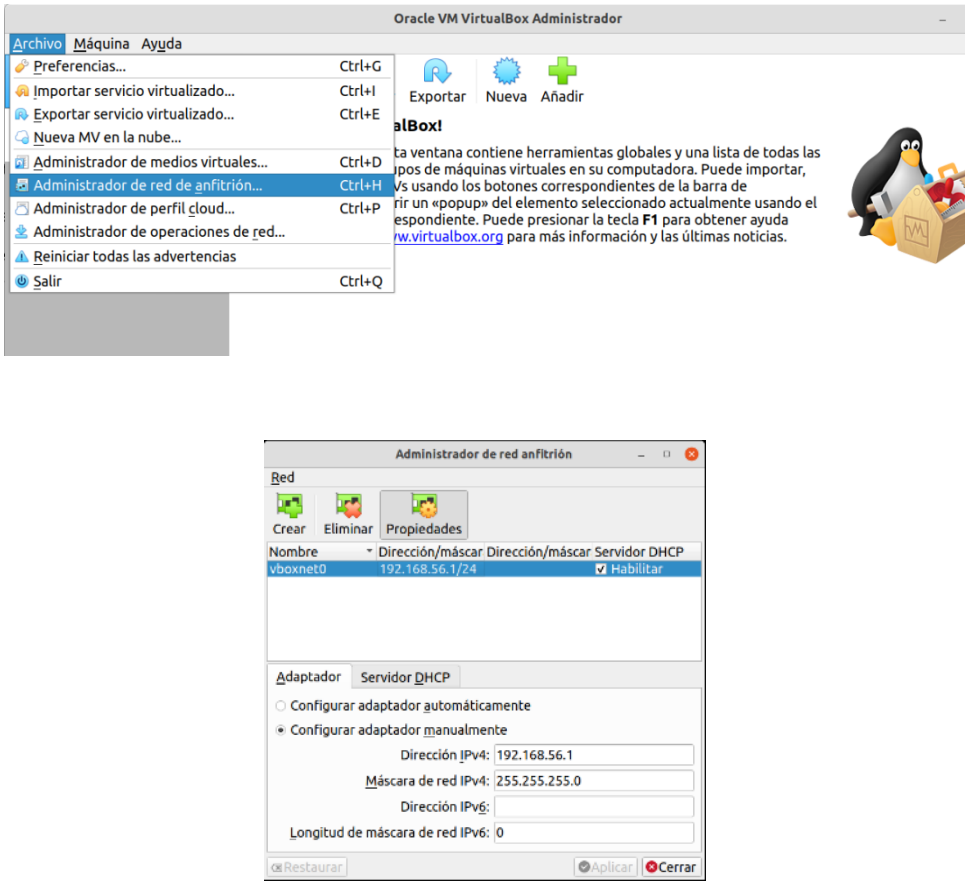


Figura 1 Creación de una red "solo anfitrión"

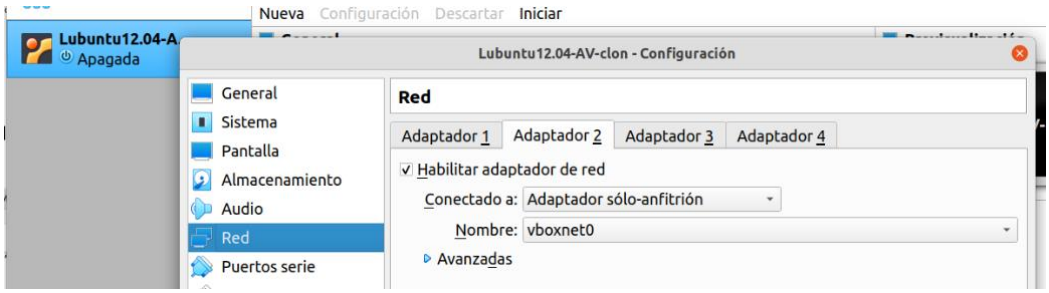


Figura 2 Configuración de una máquina virtual con una red "solo anfitrión"

Anexo IV: Cálculo de los parámetros que miden el *Slowdown*

El Slowdown[8] es una medida de cuanto más lento se ejecuta un programa en la máquina virtual con respecto a la máquina real:

$$Slowdown = \frac{t_{virtual\ (guest)}}{t_{real\ (host)}} = f_p\ N_e + (1 - f_p)$$

donde, $t_{virtual}$ es el tiempo total requerido por el programa en la máquina virtual, t_{real} el requerido en la máquina real, f_p es la fracción de tiempo que se ejecutan instrucciones privilegiadas en la ejecución sin virtualización y *Ne* el número medio de instrucciones reales que reemplazan una instrucción virtual (bien por emulación o vía *traps*).

En esta práctica se han realizado las siguientes mediciones:

tiempo de ejecución total: $t_{virtual,}\ t_{real}$

número total de instrucciones: $i_{virtual,}\ i_{real}$

El número de instrucciones sin virtualización, incluye instrucciones "inocuas" y privilegiadas. Éstas últimas se resolverán bien por *traps*, o bien por traducción binaria, multiplicando su número por el factor *Ne* antes definido. Denotando con i el número de instrucciones inocuas, e ip el número de instrucciones privilegiadas:

$$i_{real} = i + ip$$

$$i_{virtual} = i + N_e\ ip$$

En este punto, introduciremos la suposición de que el número de ciclos por instrucción, de las instrucciones privilegiadas y no privilegiadas es idéntico[9], por lo que la fracción de tiempo *f*, se puede expresar como fracción del número de instrucciones en la ejecución sin virtualización:

$$f = \frac{ip}{i + ip}$$

Combinando todas estas expresiones, obtenemos:

$$N_e = \frac{i_{real} t_{virtual} + i_{virtual} t_{real}}{i_{real} t_{real}} - 1$$

$$f = \frac{i_{real}\ t_{real}\ -\ i_{real}\ t_{virtual}}{2\ i_{real}\ t_{real}\ -\ i_{real}\ t_{virtual}\ -\ i_{virtual}\ t_{real}}$$

[8] *Slowdown = (Speedup)*$^{-1}$

[9] La herramienta *perf*, nos proporciona medidas de ciclos por lo que se podría precisar más