UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Práctica 5

Parte 1: Conceptos teóricos

1. Defina **virtualización**. Investigue cuál fue la primer implementación que se realizó.
2. ¿Qué diferencia existe entre **virtualización** y **emulación**?
3. Investigue el concepto de ***hypervisor*** y responda:
   (a) ¿Qué es un *hypervisor*?
   (b) ¿Qué beneficios traen los *hypervisors*? ¿Cómo se clasifican?
   (c) Indique por qué un *hypervisor* de tipo 1 no podría correr en una arquitectura sin tecnología de virtualización. ¿Y un *hypervisor* de tipo 2 en hardware sin tecnología de virtualización?
4. Investigue el concepto de **paravirtualización** y responda:
   (a) ¿Qué es la paravirtualización?
   (b) ¿Sería posible utilizar paravirtualización en sistemas operativos como Windows o iOS? ¿Por qué?
   (c) Mencione algún sistema que implemente paravirtualización.
   (d) Defina VMI.
   (e) ¿Qué beneficios trae con respecto al resto de los modos de virtualización?
   (f) Investigue si VMI podría correr sobre *hypervisors* de tipo 1 ó 2, y justifique por qué.
5. Investigue sobre ***containers*** en el ámbito de la virtualización y responda:
   (a) ¿Qué son?
   (b) ¿Dependen del hardware subyacente?
   (c) ¿Qué lo diferencia por sobre el resto de las tecnologías estudiadas?
   (d) Investigue qué funcionalidades son necesarias para poder implementar containers.

Parte 2: chroot, Control Groups, Namespaces y Containers

Debido a que para la realización de la práctica es necesario tener más de una terminal abierta simultáneamente tenga en cuenta la posibilidad de lograr esto mediante alguna alternativa (ssh, terminales gráficas, etc.)

chroot

En algunos casos suele ser conveniente restringir la cantidad de información a la que un proceso puede acceder. Uno de los métodos más simples para aislar servicios es `chroot`, que consiste simplemente en cambiar lo que un proceso, junto con sus hijos, consideran que es el directorio raíz, limitando de esta forma lo que pueden ver en el sistema de archivos. En esta sección de la práctica se preparará un árbol de directorios que sirva como directorio raíz para la ejecución de una *shell*.

1. Crear un subdirectorio llamado sobash dentro del directorio root. Intente ejecutar el comando *chroot /root/sobash*. ¿Cuál es el resultado? ¿Por qué se obtiene ese resultado?

2. Copiar en el directorio anterior todas las librerías que necesita el comando bash. Para obtener esta información ejecutar el comando *ldd /bin/bash*. ¿Es necesario copiar la librería linux-vdso.so.1? ¿Por qué? Dentro del directorio anterior crear las carpetas donde va el comando bash y las librerías necesarias. Probar nuevamente. ¿Qué sucede ahora?

3. ¿Puede ejecutar los comandos *cd "directorio"* o *echo*? ¿Y el comando *ls*? ¿A qué se debe esto?

4. ¿Qué muestra el comando *pwd*? ¿A qué se debe esto?

5. Salir del entorno chroot usando exit

6. Ejecute el siguiente *script* que simplemente informa al administrador datos básicos sobre el sistema operativo, lista el contenido del directorio `/home` y muestra la cantidad de procesos corriendo.

```
#!/bin/bash

while true; do
    clear
    echo -e "Hostname: $(hostname)\n"
    echo -e "Current date: $(date)\n"
    echo -e "/home directory contents:\n"
    ls -l /home
    echo -e "\nSome processes:\n$(ps -e | tail)"
    echo -e "\nProcess quantity:\n$(ps -e | wc -l)"
    sleep 10
done
```

   Este sencillo *script* es capaz de acceder a información del sistema operativo como cualquier otro servicio nativo. Esta información consiste en archivos y directorios sobre los cuales se tengan permisos de lectura, procesos en ejecución, información sobre particiones, interfaces de red, etc.

7. ¿Cuál es la finalidad de la herramienta `debootstrap`? Instalarla en un sistema operativo basado en Debian.

8. La sintaxis de `debootstrap` para crear un sistema base es: `debootstrap <suite> <target> <mirror>`. Utilice como *suite* la versión estable de Debian, `stable`, como *target* el directorio donde alojará el árbol de directorios y como *mirror* `http://httpredir.debian.org/debian`.

9. Para que un `chroot` funcione correctamente en Linux se deben montar ciertos sistemas de archivos que necesita el sistema operativo.

```
# mount --bind /dev/ target/dev/
# mount --bind /proc/ target/proc/
# mount --bind /sys/ target/sys/
```

Tenga en cuenta que debe reemplazar `target/` por el directorio donde inicializó su *debootstrap*.

10. Copie el *script* que ejecutó en la sección anterior a un directorio accesible desde el `chroot`. Por ejemplo, `target/bin/script.sh`.

11. Ejecute una *shell* cuyo directorio raíz sea `target/`. Para tal fin, utilice, con privilegios de `root`, el comando `chroot`.

12. Ejecute el *script* instalado previamente y analice los resultados.

13. ¿Puede ver los procesos que corren en el sistema operativo base? ¿Por qué?

14. Los contenidos de los directorios /home, ¿son iguales? ¿Por qué?

15. Si se ejecuta un servidor HTTP en el SO base, ¿es posible ejecutar otro servidor HTTP escuchando en el mismo puerto en el entorno chroot? ¿Por qué?

Control Groups

A continuación se probará el uso de cgroups. Para eso se crearán dos procesos que compartirán una misma CPU y cada uno la tendrá asignada un tiempo determinado.
Nota: es posible que para ejecutar xterm tenga que instalar un gestor de ventanas. Esto puede hacer con apt-get install xterm.

1. ¿Dónde se encuentran montados los cgroups? ¿Qué versiones están disponibles?

2. ¿Existe algún controlador disponible en cgroups v2? ¿Cómo puede determinarlo?

3. Analice que sucede si remueve un controlador de cgroups v1 (por ej. umount /sys/fs/cgroup/rdma).

4. Crear dos cgroups dentro del subsistema **cpu** llamados cpualta y cpubaja. Controlar que se hayan creado tales directorios y ver si tienen algún contenido

```
# mkdir /sys/fs/cgroup/cpu/"nombre_cgroup"
```

5. En base a lo realizado, ¿qué versión de cgroup se está utilizando?

6. Indicar a cada uno de los cgroups creados en el paso anterior el porcentaje máximo de CPU que cada uno puede utilizar. El valor de cpu.shares en cada cgroup es 1024. El cgroup cpualta recibirá el 70 % de CPU y cpubaja el 30 %.

```
# echo 717 > /sys/fs/cgroup/cpu/cpualta/cpu.shares
# echo 307 > /sys/fs/cgroup/cpu/cpubaja/cpu.shares
```

7. Iniciar dos sesiones por ssh a la VM.(Se necesitan dos terminales, por lo cual, también podría ser realizado con dos terminales en un entorno gráfico). Referenciaremos a una terminal como termalta y a la otra, termbaja.

8. Usando el comando taskset, que permite ligar un proceso a un core en particular, se iniciará el siguiente proceso en background. Uno en cada terminal. Observar el PID asignado al proceso que es el valor de la columna 2 de la salida del comando.

```
# taskset -c 0 md5sum /dev/urandom &
```

9. Observar el uso de la CPU por cada uno de los procesos generados (con el comando top en otra terminal). ¿Qué porcentaje de CPU obtiene cada uno aproximadamente?

10. En cada una de las terminales agregar el proceso generado en el paso anterior a uno de los cgroup (termalta agregarla en el cgroup cpualta, termbaja en cpubaja. El process_pid es el que obtuvieron después de ejecutar el comando taskset)

```
# echo "process_pid" > /sys/fs/cgroup/cpu/cpualta/cgroup.procs
```

11. Desde otra terminal observar como se comporta el uso de la CPU. ¿Qué porcentaje de CPU recibe cada uno de los procesos?

12. En termalta, eliminar el job creado (con el comando jobs ven los trabajos, con kill %1 lo eliminan. No se olviden del %.). ¿Qué sucede con el uso de la CPU?

13. Finalizar el otro proceso md5sum.

14. En este paso se agregarán a los cgroups creados los PIDs de las terminales (Importante: si se tienen que agregar los PID desde afuera de la terminal ejecute el comando echo $$ dentro de la terminal para conocer el PID a agregar. Se debe agregar el PID del shell ejecutando en la terminal).

```
# echo $$ > /sys/fs/cgroup/cpu/cpualta/cgroup.procs  (termalta)
# echo $$ > /sys/fs/cgroup/cpu/cpubaja/cgroup.procs  (termbaja)
```

15. Ejecutar nuevamente el comando *taskset -c 0 md5sum /dev/urandom &* en cada una de las terminales. ¿Qué sucede con el uso de la CPU? ¿Por qué?

16. Si en termbaja ejecuta el comando *taskset -c 0md5sum /dev/urandom &* (deben quedar 3 comandos md5 ejecutando a la vez, 2 en el termbaja). ¿Qué sucede con el uso de la CPU? ¿Por qué?

Namespaces

A continuación se crearán dos "networks namespaces" y se comunicarán entre ellos. Para esto se utilizará el comando "ip" que debe ejecutarse como root.

1. Explique el concepto de namespaces. ¿Cuáles son los posible namespaces disponibles?

2. Crear dos namespaces que se llamarán nsserver y nsclient *nsserver*:

```
# ip netns add "nombre_namespace"
```

   ip netns list: permite ver si se crearon los namespaces

3. Crear dos interfaces virtuales, tipo veth (se generan de a pares y están conectadas por medio de una tubería), que se llamarán vethsrv y vethcli:

```
# ip link add "nombre_veth0" type veth peer name "nombre_veth1"
```

ip link list: permite ver las veth recién creadas (aún pertenecen al namespace global o default)

4. A continuación se deben agregar las veth a los namespaces (vethsrv a nsserver y vethcli a nsclient)

```
# ip link set "nombre_vethX" netns "nombre_namespace"
```

ip netns exec nombre_namespace ip link list: permite ver las interfaces correspondientes a cada namespace (puede verificar que las interfaces creadas ya no se encuentran en el namespace "global")

5. Asignarle IPs a las interfaces virtuales de cada uno de los namespaces (10.10.10.1/24 a la interface en nsserver y 10.10.10.2/24 a la existente en nsclient)

```
# ip netns exec "nombre_namespace" ip addr add 10.10.10.x/24 \
  dev "nombre_vethX"
```

Utilice los comando correspondientes para comprobar que se han asignado correctamente las IPs:

```
# ip netns exec "nombre_namespace" ip addr show
```

6. Si el comando anterior indica que las interfaces están bajas (DOWN) se deben activar. Puede que también la de loopback esté down:

```
# ip netns exec "nombre_namespace" ip link set "nombre_vethX" up
```

7. ¿Es posible realizar un ping entre ambos namespaces?

8. Ejecutar el comando nc como servidor, usando el puerto 9043, en el namespace nsserver y el client en el nsclient. Comprobar que funciona en forma correcta.

9. Sin cerrar los comandos nc, ejecutar en el entorno *global* el comando *ss -nat*. ¿Puede ver el estado en que se encuentra el puerto 9043? ¿Por qué? Ejecute el mismo comando pero con la opción *-N "namespace"*. ¿Es posible ver alguna información con respecto a ese puerto?

10. Ejecutar un bash dentro de uno de los nameservers (ip netns exec nsservidor bash) y ver qué interfaces de red están activas. ¿Se ven las mismas inferfaces que fuera del namespace?

Containers

En GNU/Linux hay varias implementaciones de la tecnología de containers que provee el kernel. En esta sección de la práctica estudiaremos una de ellas, **LXC: Linux Containers**.

1. Instalar LXC mediante el comando:

```
# apt-get install lxc
```

2. Comprobar mediante el siguiente comando si el *kernel* soporta LXC:

```
# lxc-checkconfig
```

Nota: puede que no todas las opciones estén habilitadas

3. Comprobar los *templates* disponibles (a partir de donde se crean los *containers*):

```
# ls /usr/share/lxc/templates
```

4. Crear un nuevo *container* desde el *template* de Debian con el nombre `sodebian`:

```
# lxc-create -t debian -n sodebian
```

Nota: dependiendo del template seleccionado durante la instalación se muestra el usuario y su contraseña, generada aleatoriamente que se utilizará para ingresar al *container*. Este comando puede demorar varios minutos la primera vez que es ejecutado ya que debe descargar muchos componentes.

5. Una vez creado el *container* iniciarlo mediante el siguiente comando:

```
# lxc-start -n sodebian
```

Nota: `-n` para ingresar el nombre del *container* que queremos iniciar. Con la secuencia CTRL-A-Q es posible salir del *container*

6. Utilizar el comando `lxc-ls -f` para comprobar el estado de los *containers*. ¿En qué estado se encuentra el *container* `sodebian`?

7. Mediante el siguiente comando ingresar al *container*

```
# lxc-console -n sodebian -t 0
```

8. ¿Tiene la password para ingresar *container*? En caso de no tenerla modificar la password de root usando *chroot*
Nota: con el comando lxc-attach es posible acceder a un *container* sin tener la password

9. Ejecutar el *script* utilizado en los ejercicios anteriores y responder:
   (a) ¿Puede acceder al `/home` del sistema operativo base?
   (b) ¿Es posible ver los procesos que están corriendo en el sistema operativo base?

10. Montar el *home directory* del usuario con el que ingresó al SO base en el directorio /mnt del contenedor
    (a) ¿Se montó correctamente el directorio? (Compare el contenido de ambos directorios)
    (b) En el directorio /mnt del *container* crear una carpeta MountSO. ¿Existe también en el *home directory*?
    (c) Elimine la carpeta creada en el punto anterior ubicado en *home directory* y comprobar si también se elimina en el directorio del contenedor
    (d) Desmontar el directorio

11. ¿Qué interfaces de red existen en el *container*? Analizar el contenido del archivo config en el directorio `/var/lib/lxc/sodebian/`.

12. Usando el archivo config del container asignarle una nueva interface de red con la dirección 172.16.2.10/24. La interface debe ser de tipo veth y asociarse en el otro extremo a un bridge llamado brso1 (tengan en cuenta que el bridge definido de esta manera se pierde al rebootear):

(a) brctl addbr brso1 - crea un bridge virtual en la VM

(b) brctl show - para ver los bridges definidos en la VM

(c) En el archivo config del container agregar las siguientes líneas:

```
## Se indica el tipo de interface (veth igual al tipo de interface visto en
lxc.net.0.type=veth
## Un extremo de la interface va en el container y el otro en el bridge brso
lxc.net.0.link=brso1
## Levantamos la interface
lxc.net.0.flags = up
## Definimos la dirección MAC
lxc.net.0.hwaddr = 4a:49:43:49:79:bf
## Definimos la dirección IPv4
lxc.net.0.ipv4.address = 172.16.2.10/24
```

13. Reinicar el container y utilizando el comando *ip addr show* comprobar si todo está correcto.

14. Detener el *container* llamado `sodebian`.

```
# lxc-stop -n sodebian
```

15. Clonar el *container* utilizando el comando `lxc-copy`. Llamar so1 al nuevo *container*.

```
# lxc-copy -n "old_container" -N "new_container"
```

16. En estas condiciones, ¿es posible iniciar ambos *containers*? ¿Qué debería hacer para solucionarlo?

17. Iniciar los dos *containers* y ejecutar nuevamente `lxc-ls -f`. ¿Ambos *containers* tiene IP asignada?

18. Al loguear en el nuevo *container*, ¿qué nombre tiene? ¿Podría ser modificado ese nombre sin estar en la consola del *container*? De ser posible, cambiarlo a so1.

19. ¿Es posible hacer un ping entre ambos *containers*? ¿Por qué?

20. Para solucionar el punto anterior vamos a conectar ambos *containers* a Internet para que puedan instalarse el paquete iputils-ping:

    (a) ip addr del 10.0.2.15/24 dev enp0s3 –removemos la IP de la interface enp0s3, que en esta VM es la que nos da la salida a Internet

    (b) brctl addif brso1 enp0s3 – agregamos la interface al mismo bridge donde están conectados los *containers*

    (c) dhclient brso1 – asignamos IP al bridge (la interface enp0s3 debe quedar sin IP)

    (d) Remover del archivo config de cada *container* las líneas donde se definen las IPs

    (e) Reiniciar los *containers*. Ver si tomaron IP del mismo rango que el bridge (debería ser de la red 10.0.2.0/24)

    (f) Ingresar al *container* sodebian e instalar el paquete requerido

    (g) Probar de hacer ping al otro *container* o al nodo host

21. Crear un nuevo contenedor, llamado so2, igual que como se generó sodebian. ¿Por qué tarda mucho menos tiempo que cuando se creó sodebian?

22. Usando los cgroups limitar la cantidad máxima de memoria RAM que puede consumir un *container*. Ver cuánta memoria tiene asignada (comandos top o free) y bajarla a la mitad. Por ej. si tiene asignados 1GB quedaría así (lo llevamos a 512MB):

```
# lxc-cgroup -n "nombre_contenedor" memory.limit_in_bytes 536870912
```

Nota: el *container* debe estar corriendo

23. Comprobar que se modificó la memoria RAM en el *container*.

24. ¿Dónde se debería ver reflejado la modificación anterior? ¿Se ve realmente el cambio realizado en el lugar correspondiente?

25. Volver la RAM a su valor original (1GB en este ejemplo)

26. Eliminar definitivamente todos los *containers* creados mediante el comando:

```
# lxc-destroy -n "nombre_contenedor"
```