Sistemas Operativos

Práctica 4

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*? Investigue cuándo se realizó la primer implementación de esta tecnología.
   (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 del *kernel* Linux permiten la implementación de *containers*.

Parte 2: *Control Groups, Namespaces y Containers*

`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. 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: $(date)\n"
    echo –e "/home directory contents:\n"
    ls –l /etc
    echo –e "\nSome processes:\n$(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.

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

6. 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`.

7. 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*.

8. Copie el *script* que ejecutó en la sección anterior a un directorio accesible desde el `chroot`. Por ejemplo, `target/bin/script.sh`.
9. Ejecute una *shell* cuyo directorio raíz sea `target/`. Para tal fin, utilice, con privilegios de `root`, el comando `chroot`.
10. Ejecute el *script* instalado previamente y analice los resultados.
11. ¿Puede ver los procesos que corren en el sistema operativo base? ¿Por qué?
12. Los contenidos de los directorios /home, ¿son iguales? ¿Por qué?
13. ¿Qué desventajas encuentra en el uso de `chroot` como medida de seguridad?
14. 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.

1. Crear dos grupos dentro del subsistema **cpu** llamadas cpualta y cpubaja. Controlar que se hayan creado tales directorios y ver si tiene algún contenido

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

2. 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
```

3. Usando el comando taskset, que permite ligar un proceso a un core en particular, se iniciarán dos procesos en background. (Puede suceder que estos comandos no puedan ejecutar como root. Debe hacerlo como un usuario normal)

```
# taskset -c 0 xterm -bg blue &
# taskset -c 0 xterm -bg red &
```

4. En cada una de las xterm generadas ejectuar el comando como un job. Observar el uso de la CPU por cada uno de los procesos generados (con el comando top en otra terminal)

```
# md5sum /dev/urandom &
```

5. En cada una de las xterm agregar el proceso generado en el paso anterior a uno de los cgroup (xterm con el fondo azul en el cgroup cpualta)

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

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

7. Finalizar los dos procesos md5sum.

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

```
# echo $$ > /sys/fs/cgroup/cpu/cpualta/cgroup.procs
# echo $$ > /sys/fs/cgroup/cpu/cpubaja/cgroup.procs
```

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

10. Si en la xterm red ejecuta el comando *md5sum /dev/urandom &* (deben quedar 3 comandos md5 ejecutando a la vez, 2 en el xterm rojo). ¿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 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:

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

7. Ejecutar el comando nc como servidor, usando el puerto 9043, en el namespace nsserver y el client en en el nsclient. Comprobar que funciona en forma correcta.
8. 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?
9. Ejecutar un bash dentro de uno de los nameservers y ver qué interfaces de red están activos

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
```

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. Modificar el nombre del contenedor sin estar en su consola.

11. 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 Mount2018. ¿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

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

13. Configurar una interface en el *container* con una dirección asignada por DHCP. Para esto se usará el servicio lxc-net (debería haberse instalado junto con el lxc) y un bridge virtual (puede ser necesario instalar el paquete bridge-utils):
    (a) Si no existe crear el archivo lxc-net en el directorio /etc/default con el siguiente contenido:

```
USE_LXC_BRIDGE="true"
LXC_BRIDGE="lxcbr0"
LXC_ADDR="10.10.10.1"
LXC_NETMASK="255.255.255.0"
LXC_NETWORK="10.10.10.0/24"
LXC_DHCP_RANGE="10.10.10.2,10.10.10.250"
LXC_DHCP_MAX="249"
LXC_DHCP_CONFILE=""
LXC_DOMAIN=""
```

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

```
lxc.network.type = veth    # Tipo de interface de red a crear
lxc.network.link = lxcbr0  # Dispositivo al que se conecta esta interface
```

    Nota: si existe lxc.network.type = empty debe eliminarla o comentarla
    (c) Iniciar, o reiniciar, el servicio lxc-net (systemctl stop/start/restart lxc-net)

(d) Reiniciar el *container* y ver si se le asignó una dirección IP

(e) Con los comandos *ip addr show* y *brctl show* puede ver las interfaces del contenedor y como se agregaron al bridge

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

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

15. Clonar el *container* utilizando el comando `lxc-copy`.

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

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

17. Crear un nuevo contenedor, llamado so2, igual que como se generó sodebian

18. Iniciarlo y ver si se le asignó dinámicamente una dirección IP.

19. 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 2GB quedaría así:

```
# lxc-cgroup -n "nombre_contenedor" memory.limit_in_bytes 1073741824
```

Nota: el *container* debe estar corriendo

20. Comprobar que se modificó la memoria RAM en el *container*. Volverlo a su valor original (2GB en este ejemplo)

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

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

22. Crear dos containers a partir de lo indicado en un archivo:

(a) La IP de los containers debe pertenecer a la red 192.168.100.0/24

(b) Ambos contenedores deben conectarse a un bridge virtual llamada brSO

(c) Probar si es posible alcanzar al otro container (ping IP)