Sistemas Operativos - Práctica 4A [2025]

Cgroups & Namespaces

Parte 1: Conceptos teóricos

1. Defina **virtualización**. Investigue cuál fue la primera implementación que se realizó.

La virtualización permite dividir los recursos de hardware de un sistema —como procesadores, memoria y almacenamiento, entre otros— entre varios sistemas virtuales, denominados máquinas virtuales (VM). Utiliza software para crear una capa de abstracción sobre el hardware del sistema. Cada VM ejecuta su propio sistema operativo y se comporta como un ordenador independiente, aunque se esté ejecutando en una parte del hardware del sistema subyacente real.

El nacimiento de la virtualización se remonta a 1964, cuando IBM diseñó e introdujo CP-40, un proyecto de investigación experimental de tiempo compartido para el sistema/360 de IBM. El CP-40, que más tarde evolucionó hasta convertirse en el CP-67 y luego en Unix, proporcionó un hardware informático capaz de soportar múltiples usuarios simultáneos y sentó las bases de las máquinas virtuales. ^1

2. ¿Qué diferencia existe entre virtualización y emulación? Un emulador permite que un entorno informático replique la funcionalidad de otro, permitiendo que el software escrito para una plataforma se ejecute en otra^2. Los emuladores funcionan convirtiendo instrucciones destinadas a un sistema en un formato que otro sistema pueda ejecutar. Esto suele implicar:

- Traducción binaria (conversión de código de máquina de una arquitectura a otra).
- Virtualización (ejecución de software en un entorno controlado y aislado).
- Abstracción de hardware (permitir que el software interactúe con una configuración de hardware diferente).

> La emulación implica simular el hardware y el software de un sistema para que un programa se ejecute en otro, mientras que la virtualización crea entornos virtuales que imitan máquinas separadas en el mismo hardware

3. Investigue el concepto de hypervisor y responda ^3: (a) ¿Qué es un hypervisor? Un **hipervisor** es la capa de software que coordina las máquinas virtuales. Sirve como interfaz entre la VM y el hardware físico subyacente, y garantiza que cada uno tenga acceso a los recursos físicos que necesita ejecutar. También se asegura de que las máquinas virtuales no interfieran entre ellas afectando al espacio de memoria o los ciclos de cálculo del resto.

(b) ¿Qué beneficios traen los hypervisors? ¿Cómo se clasifican? ^4 **Principales beneficios**:

- Eficiencia: permite que varias áquinas vrituales utilicen el mismo hardware físico
- Escalabilidad: crean, implementan y retiran maquinas virtuales.
- Ahorro de costos: permiten ejecutar varias áquinas virtuales en una sola áquina física.
- Seguridad: aíslan las VM entre sí y la de máquina host. Hay dos tipos de hipervisores:
- **Hipervisores de tipo 1 o "bare-metal"**: interactúan con los recursos físicos subyacentes, sustituyendo por completo al sistema operativo tradicional. Son muy eficientes porque acceden directamente al hardware físico.

- **Hipervisores de tipo 2**: se ejecutan como una aplicación en un sistema operativo existente. Normalmente se utilizan en dispositivos de punto final para ejecutar sistemas operativos alternativos e implican una sobrecarga de rendimiento porque deben utilizar el sistema operativo del host para acceder y coordinar los recursos de hardware subyacentes. Como accede a los recursos informáticos, de memoria y de red a través del sistema operativo host, introduce problemas de latencia que pueden afectar el rendimiento.

4. ¿Qué es la **full virtualization**? ¿Y la **virtualización asistida por hardware**? La full virtualization es una técnica de virtualización donde el hipervisor crea una VM completamente idéntica al hardware físico, de manera que el sistema operativo invitado no necesita ser modificado en absoluto. Por lo tanto el sistema operativo *invitado* cree que está corriendo en una máquina física real. Como el sistema invitado no puede acceder directamente al hardware, el hipervisor intercepta y emula ciertas instrucciones que requieren privilegios (como acceso a dispositivos o controladores). Esto lo lograba mediante traducción binaria dinámica: el hipervisor analizaba y convertía las instrucciones privilegiadas del sistema invitado en otras seguras que no causaran conflictos con el sistema anfitrión.

La virtualización asistida por hardware es una tecnología integrada en procesadores modernos que facilita la virtualización permitiendo ejecutar instrucciones privilegiadas directamente en la CPU sin necesidad de emulación compleja. Entonces, el hipervisor peude delegar al hardware la gestión de tareas: manejo de cambios de contexto e ntre invitados y hosts, control de interrupciones y acceso a memoria, y protección del aislamiento entre VMs.

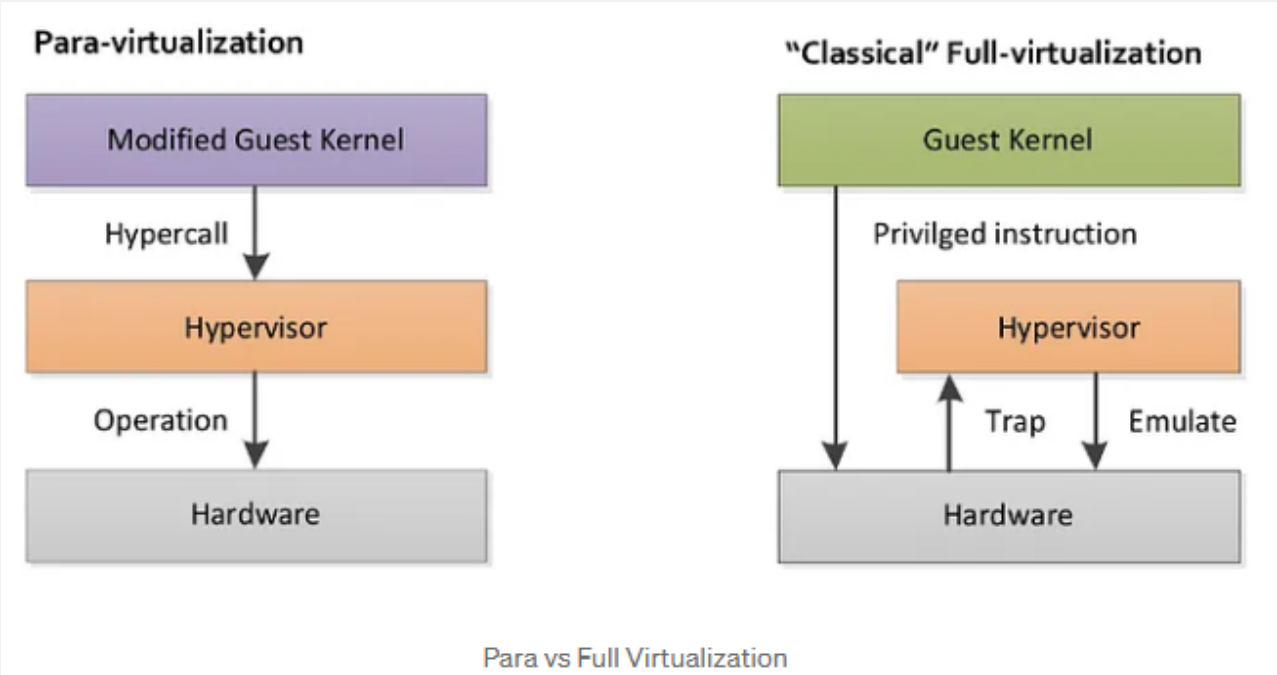
> La virtualización completa puede implementarse con o sin soporte de hardware: la virtualización completa antes se realizaba mediante software (vinary trasnlation) y actualmente los hipervisores modernos usan la virtualización asistida por harware.

5. ¿Qué implica la técnica **binary translation**? ¿Y **trap-and-emulate**?

- **Binary translation**: el hipervisor intercepta y traduce dinámicamente instrucciones del código del sistema operativo invitado y las reemplaza por secuencias seguras que no comprometen la estabilidad del host. Funcionamiento:
    - El SO guest intenta ejecutar una instrucción privilegiada.
    - El hipervisor analiza bloques de código binario del invitado, los traduce a un equivalente seguro y los ejecuta.
- **Trap and emulate**: el hipervisor permite que el sistema operativo invitado intente ejecutar instrucciones privilegiados, pero cuando lo hace, la CPU genera una trap y cede el control al hipervisor, que luego emula el comportaiento de sea instrucción. Funcionamiento:
    - El SO guest ejecuta una instrucción privilegiada.
    - Como no tiene privilegios, la CPU lanza una excepción.
    - El hipervisor intercepta la excepción y emula la instrucción en software.
    - Devuelve el control al invitado.

> Tanto binary translation como trap-and-emulate son técnicas que se usan para implementar virtualización completa (full virtualization) cuando el sistema invitado intenta ejecutar instrucciones

> privilegiadas, y no se cuenta con soporte de hardware o se busca optimizar el rendimiento.



Para vs Full Virtualization

6. Investigue el concepto de paravirtualización y responda: (a) ¿Qué es la paravirtualización? En este caso, la paravirtualización es una técnica donde el sistea operativo invitado es modificado explícitamente para que colabore con el hipervisor. En lugar de intentar ejecutar instrucciones privilegiadas directamente, el SO llama al hipervisor usando una interfaz definida (*hypercalls*). Paravirtualization eliminates the need for hardware emulation, resulting in lower overhead and improved performance compared to full virtualization. (b) Mencione algún sistema que implemente paravirtualización. Examples of para virtualization include Xen hypervisor with Xen-aware guest operating systems.[^5] (c) ¿Qué beneficios trae con respecto al resto de los modos de virtualización?

- **Menor overhead**: el sistema operativo invitado colabora con el hipervisor, usando hypercalls en lugar de ejecutar instrucciones que deben interceptarse o traducirse.
- **Mejor aprovechamiento de recursos**: al eliminar parte de la complejidad del monitoreo constante del invitado, el hipervisor consume menos CPU y memoria.

> La paravirtualización sacrifica compatibilidad a cambio de rendimiento. Es ideal en entornos donde se pueden controlar los sistemas operativos invitados (como en nubes privadas, data centers, o sistemas Linux optimizados).

7. Investigue sobre containers y responda[^6]: (a) ¿Qué son? Son unidades ejecutables de software que empaquetan el código de la aplicación en conjunto con sus bibliotecas y dependencias. Permite que el código se ejecute en cualquier entorno informático. Los contenedores aprovechan una forma de virtualización del sistema operativo (SO) en la que las características del kernel del SO (namespaces y cgroups por ejemplo)se pueden usar para aislar procesos y controlar la cantidad de CPU, memoria y disco a los que esos procesos pueden acceder.

(b) ¿Dependen del hardware subyacente? En lugar de virtualizar el hardware subyacente, los contenedores virtualizan el sistema operativo, de modo que cada contenedor contiene únicamente la aplicación y sus biblitoecas, archivos de configuración y dependencias.

(c) ¿Qué lo diferencia por sobre el resto de las tecnologías estudiadas?

- Virtualización crea máquinas virtuales completas, cada una con su propio sistema operativo (kernel, drivers, etc.).
- Contenedores comparten el mismo kernel del sistema operativo anfitrión, pero aíslan las aplicaciones y sus dependencias en entornos separados.

(d) Investigue qué funcionalidades son necesarias para poder implementar containers.

- **Namespaces**: para aislar recursos del sistema como si fuera un entorno separado. Un contenedor puede tener su propio hostname, tabla de procesos, y red, sin ver la del host.
- **Control groups**: Permiten limitar y monitorizar el uso de recursos (CPU, RAM, disco, red, etc.) por grupo de procesos. Se pueden asignar límites y prioridades. Impiden que un contenedor acapare todo el hardware.
- **Sistema de archivos**: Permiten combinar capas de archivos de solo lectura y escritura. Esto es clave para: compartir capas comunes entre contenedores (por ejemplo, la capa base de una imagen) o crear nuevos contenedores sin duplicar todo el sistema de archivos.
- `chroot` **y** `pivot_root`: Herramientas para cambiar el root filesystem de un proceso, de modo que parezca que el contenedor vive en su propio sistema de archivos aislado.

Parte 2: `chroot`, Control Groups y Namespaces

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. ¿Qué es el comando `chroot`? ¿cuál es su finalidad? El comando cambia el directorio raíz al directorio especificado por el directorio pasado por parámetro. De este modo se puede crear un nuevo entorno separado lógicamente del directorio raíz del sistema principal. Un programa que se ejecuta en este entorno modificado no puede acceder a los archivos y comandos fuera de ese árbol de directorios del entorno.

2. 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?

```
root@so:~/sobash# chroot /root/sobash
chroot: failed to run command ‘/bin/bash’: No such file or directory
```

Se obtiene ese resultado porque después del comando, el proceso pierde acceso al sistema real y todo lo que intente ejecutar debe existir dentro del nuevo root. Dado que está vacío, no hay nada que ejecutar.

3. Cree la siguiente jerarquía de directorios dentro de `sobash`:

```
sobash/
├── bin
```

```
├── lib
│   └── x86_64-linux-gnu
└── lib64
```

4. Verifique qué bibliotecas compartidas utiliza el binario `/bin/bash` usando el comando `ldd /bin/bash`. ¿En qué directorio se encuentra `linux-vdso.so.1`? ¿Por qué?

```
root@so:~/sobash# ldd /bin/bash
        linux-vdso.so.1 (0x00007f52d82bf000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007f52d8140000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f52d7f5f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f52d82c1000)
```

`linux-vdso.so.1` es un objeto virtual compartido que no tiene ningún archivo físico en el disco; es una parte del kernel que se exporta al espacio de direcciones de cada programa cuando se carga. El mecanismo vDSO (Objeto Dinámico Virtual Compartido) permite al kernel exportar un segmento de memoria al espacio de direcciones de un proceso. Esto permite que llamadas específicas del sistema se ejecuten en el espacio de usuario en lugar del espacio del kernel, lo que ofrece una ventaja de rendimiento al evitar la sobrecarga de un cambio de contexto.

5. Copie en `/root/sobash` el programa `/bin/bash` y todas las librerías utilizadas por el programa bash en los directorios correspondientes. Ejecute nuevamente el comando chroot ¿Qué sucede ahora?

```
#Sacado de chatgpt pero usé lo mismo
sudo cp -v /lib/x86_64-linux-gnu/libtinfo.so.6 /root/sobash/lib/x86_64-linux-gnu/
sudo cp -v /lib/x86_64-linux-gnu/libc.so.6 /root/sobash/lib/x86_64-linux-gnu/
sudo cp -v /lib64/ld-linux-x86-64.so.2 /root/sobash/lib64/
```

Al ejecutar de nuevo el comando se tiene:

```
root@so:~/sobash# chroot /root/sobash
bash-5.2# echo "Hola mundo"
Hola mundo
```

Se puedo entrar al entorno.

6. ¿Puede ejecutar los comandos `cd "directorio"` o `echo`? ¿Y el comando `ls`? ¿A qué se debe esto? Tanto `cd` como `echo` funcionan por que el shell lo iplementa directamente. En el caso de `ls` no porque es un programa externo y no está el binario en el entorno, entonces es necesario copiarlo dentro del entorno (igual a como se hizo con bash).

7. ¿Qué muestra el comando `pwd`? ¿A qué se debe esto?

```
bash-5.2# pwd
/
```

Se debe a que nos encontramos dentro del entorno `chroot`.

8. Salir del entorno chroot usando exit

```
bash-5.2# exit
exit
root@so:~/sobash#
```

9. Usando el repositorio de la cátedra acceda a los materiales en `practica4/02-chroot`: a. Verifique que tiene instalado `busybox` en `/bin/busybox` b. Cree un `chroot` con busybox usando `/buildbusyboxroot.sh`

```
root@so:/home/so/practica4/codigo-para-practicas/practica4/02-chroot# make
./buildbusyboxroot.sh
        linux-vdso.so.1 (0x00007f669f1bb000)
        libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2
(0x00007f669f0e0000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f669eeff000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f669f1bd000)
BusyBox root filesystem created in /home/so/practica4/codigo-para-
practicas/practica4/02-chroot/busyboxroot
You can now chroot into it with:
chroot /home/so/practica4/codigo-para-practicas/practica4/02-chroot/busyboxroot
/bin/sh
```

c. Entre en el `chroot`

```
root@so:/home/so/practica4/codigo-para-practicas/practica4/02-chroot# chroot
/home/so/practica4/codigo-para-practicas/practica4/02-chroot/busyboxroot /bin/sh


BusyBox v1.35.0 (Debian 1:1.35.0-4+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ #
```

d. Busque el directorio `/home/so` ¿Qué sucede? ¿Por qué?

e. Ejecute el comando `ps aux` ¿Qué procesos ve? ¿Por qué (pista: ver el contenido de `/proc`)?

```
/ # ps aux
PID   USER     COMMAND
```

No se ve ná xd.

```
/ # cd proc/
/proc # ls
/proc #
```

Como `chroot` aísla completamente el sistema de archivos entonces si dentro de este entorno no se encuentra el directorio entonces no puede ser accedido.

f. Monte `/proc` con `mount -t proc proc /proc` y vuelva a ejecutar `ps aux` ¿Qué procesos ve? ¿Por qué?

```
/ # mount -t proc proc /proc
/ # ps aux
PID   USER     COMMAND
    1 0        {systemd} /sbin/init
    2 0        [kthreadd]
    3 0        [pool_workqueue_]
    4 0        [kworker/R-rcu_g]
    5 0        [kworker/R-sync_]
    # Más y más procesos...
 8262 0        /bin/sh
 8677 0        [kworker/1:0-eve]
 8734 1000     /bin/bash --init-file /home/so/.vscode-server/cli/servers/Stable-
17baf841131aa23349f217ca7c570c76ee87b957/server/out/vs/workbench/contrib/
 8768 1000     sleep 180
 9146 0        ps aux
```

Luego de montar `/proc` el comando `ps aux` muestra todos los procesos del sistema real (no solo del entorno) porque al montarlo el chroot puede acceder al contenido real de `/proc` que sigue siendo el del sistema host.

g. Acceda a `/proc/1/root/home/so` ¿Qué sucede?

```
/ # cd /proc/1/root/home/so
sh: getcwd: No such file or directory
(unknown) # ls
install_deps.sh  kernel           practica1        practica2        practica3
practica4
sh: getcwd: No such file or directory
(unknown) #
```

h. ¿Qué conclusiones puede sacar sobre el nivel de aislamiento provisto por `chroot`? Incluso desde dentro de un `chroot`, si se tienen permisos para acceder a `/proc`, se puede usar `/proc/1/root/...` para salir del aislamiento del `chroot` y ver el sistema de archivos real. Por eso, los chroot no son una verdadera medida de seguridad: un usuario con privilegios root dentro del chroot puede escapar fácilmente usando `/proc`.

Control Groups

Preparación

Actualmente Debian y la mayoría de las distribuciones usan CGroups 2 por defecto, pero para esta práctica usaremos CGroups 1. Para esto es necesario cambiar un parámetro de arranque del sistema en grub:

1. Editar /etc/default/grub:

```
# Cambiar:
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
# Por:
GRUB_CMDLINE_LINUX_DEFAULT="quiet systemd.unified_cgroup_hierarchy=0"
```

2. Actualizar la configuracion de GRUB
3. Reiniciar la máquina
4. Verificar que se esté usando CGroups 1. Para esto basta con hacer `ls /sys/fs/cgroup/`. Se deberían ver varios subdirectorios como cpu, memory, blkio, etc. (en vez de todo montado de forma unificada).

```
so@so:/etc$ ls /sys/fs/cgroup
blkio  cpuacct      devices  hugetlb  net_cls          net_prio    pids  systemd
cpu    cpu,cpuacct  freezer  misc     net_cls,net_prio  perf_event  rdma  unified
```

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`.

Taller

1. ¿Dónde se encuentran montados los cgroups? ¿Qué versiones están disponibles? En `/sys/fs/cgroup` se encuentra el directorio que contendrá los puntos de montaje de las diferentes jerarquías. Cada controlador se monta en su propio sistema de archivos, por ejemplo:

```
/sys/fs/cgroup/cpu/
/sys/fs/cgroup/memory/
```

Hay dos versiones Cgroups v1 y Cgroups v2:

Cgroups v1	Cgroups v2

Cgroups v1	Cgroups v2
Cada controlador (CPU, memoria, etc.) tiene su propio montaje independiente.	Usa un sistema unificado de jerarquía: todos los controladores se montan en un solo árbol.
Usa varios archivos por controlador (`memory.limit_in_bytes`, `cpu.shares`, etc.).	Usa una interfaz estandarizada: por ejemplo, en lugar de `memory.limit_in_bytes`, usa `memory.max`.
Se pueden añadir procesos a cualquier nivel del árbol, incluso a nodos intermedios. Puede egnerar ambigüedad en el control de recursos.	Sólo se pueden añadir procesos a **nodos hoja**.

A tener en cuenta:

- **Cgroup v1**: múltiples montajes por controlador en `/sys/fs/cgroup/<controlador>/`.
- **Cgroup v2**: un único montaje unificado en `/sys/fs/cgroup/`, con archivo `cgroup.controllers`.

2. ¿Existe algún controlador disponible en cgroups v2? ¿Cómo puede determinarlo? Sí, cgroups v2 tiene sus propios controladores. Mediante `cat /sys/fs/cgroup/cgroup.controllers` se pueden consultar los controlares disponibles.

3. ¿Qué sucede si se remueve un controlador de cgroups v1 (por ej. `umount /sys/fs/cgroup/rdma`). El kernel deja de aplicar los controles asociados a ese subsistema y tampoco se podrán crear nuevos cgroups que usen ese controlador.

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>
```

Resultado:

```
root@so:/sys/fs/cgroup# mkdir /sys/fs/cgroup/cpu/cpubaja
root@so:/sys/fs/cgroup# ls /sys/fs/cgroup/cpu/cpubaja
cgroup.clone_children      cpu.cfs_burst_us
cgroup.procs               cpu.cfs_period_us
cpuacct.stat               cpu.cfs_quota_us
cpuacct.usage              cpu.idle
cpuacct.usage_all          cpu.shares
cpuacct.usage_percpu       cpu.stat
cpuacct.usage_percpu_sys   cpu.stat.local
cpuacct.usage_percpu_user  notify_on_release
cpuacct.usage_sys          tasks
cpuacct.usage_user
#Se imprime lo mismo para el caso de cpualta
```

5. En base a lo realizado, ¿qué versión de cgroup se está utilizando? Se está usando la v1 dado que un recurso (cpu) le generamos un cgroup. El hecho de que exista `/sys/fs/cgroup/cpu/` indica que cpu está montado de forma independiente, como es típico de v1.

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 &
```

```
root@so:/sys/fs/cgroup# taskset -c 0 md5sum /dev/u
random &
[1] 3217
```

```
root@so:/sys/fs/cgroup# taskset -c 0 md5sum /dev/u
random &
[1] 3250
```

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?

Resultado:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 3217 root      20   0    5476   1812   1684 R  50,3   0,1   0:32.53 md5sum
 3250 root      20   0    5476   1900   1772 R  49,7   0,1   0:26.82 md5sum
```

Se puede ver la asignación de ambos es del 50% para ambos.

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
```

Resultado:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 3217 root      20   0    5476   1812   1684 R  71,4   0,1   3:16.48 md5sum
 3250 root      20   0    5476   1900   1772 R  28,6   0,1   3:03.30 md5sum
```

Se ve aplicada la configuración del límite de CPU que puede usar tanto cpualta como cpubaja.

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?

```
root@so:/sys/fs/cgroup# jobs
[1]+  Ejecutando              taskset -c 0 md5sum /dev/urandom &
root@so:/sys/fs/cgroup# kill %1
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 3250 root      20   0    5476   1900   1772 R 100,0   0,1   5:25.48 md5sum
```

Se observa que el proceso que queda de termalta (cpualta) toma todo el CPU (100%) cuando en realidad está configurada solo para el 30%.

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?

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4239 root      20   0    5476   1752   1624 R  69,8   0,1   3:39.77 md5sum
 4311 root      20   0    5476   1752   1624 R  30,2   0,1   1:30.36 md5sum
```

A pesar de que en el paso 11 se pudo ver el mismo resultado acá el camino es diferente. En este caso se agrega la terminal shell al cgroup, ésta terminal genera un nuevo proceso y acá viene la magia: este proceso hereda el cgroup del proceso padre, entonces también usa el 70% y 30% correspondiente a cada terminal.

> Una vez que el shell pertenece a un cgroup, cualquier proceso lanzado desde él también lo hace, lo cual evita tener que asignarlos manualmente.

16. Si en termbaja ejecuta el comando: `taskset -c 0 md5sum /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? Se observa que los dos procesos pertenecientes termbaja se distribuyen (compiten) los recursos entre ellos mas no ocupan el % que le corresponde del cgroup.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4239 root      20   0    5476   1752   1624 R  70,1   0,1   6:50.40 md5sum
 4311 root      20   0    5476   1752   1624 R  15,9   0,1   2:43.77 md5sum
 4446 root      20   0    5476   1756   1628 R  14,0   0,1   0:08.19 md5sum
```

Namespaces

1. Explique el concepto de namespaces. Un *namespace* envuelve un recurso global del sistema en una abstracción que hace parecer a los procesos dentro de él que tienen su propia instancia aislada del recurso global. Los cambios en un determinado recurso global sólo son visibles para los procesos de ese espacio de nombres y no afectan al resto del sistema ni a otros espacios de nombres [^8].

> Cada namespace crea una vista limitada o independiente de algún recurso del sistema, como procesos, red o sistema de archivos.

2. ¿Cuáles son los posibles namespaces disponibles?

Namespace	Descripción
`mnt`	Aísla los puntos de montaje (sistema de archivos). Permite que diferentes procesos tengan su propio conjunto de montajes.
`pid`	Aísla los identificadores de procesos. Los procesos en un namespace ven solo sus propios procesos (útil para containers).
`net`	Aísla interfaces de red, direcciones IP, rutas, puertos. Usado para crear redes virtuales por contenedor.
`ipc`	Aísla recursos de comunicación entre procesos (como colas de mensajes, semáforos, memoria compartida).
`uts`	Aísla el nombre del sistema y el dominio (hostname, nodename).
`user`	Aísla el espacio de IDs de usuario. Permite mapear un usuario sin privilegios fuera del namespace como `root` dentro.
`cgroup`	Aísla la vista del árbol de controladores de cgroups. Procesos pueden ver/controlar solo su jerarquía.
`time`	Aísla el reloj del sistema (desde Linux 5.6). Permite tener distintos relojes (por ej. para simulaciones).

3. ¿Cuáles son los namespaces de tipo Net, IPC y UTS una vez que inicie el sistema (los que se iniciaron la ejecutar la VM de la cátedra)? Se analizó el proceso con PID 1 mediante el uso de `ls -l /proc/1/ns` y se obtiene lo siguiente:

```
lrwxrwxrwx 1 root root 0 may 10 22:39 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 may 10 22:39 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 may 10 22:39 uts -> 'uts:[4026531838]'
```

El identificador de namespace es generado por el kernel para distinguir entre diferentes instancias del tipo de namespace. Cada entrada es un enlace simbólico a una instancia del namespace del tipo correspondiente.

> Dado el ID de un namespace, si otro proceso tiene el mismo entonces éste comparte ese namespace. Si tiene uno distinto, está aislado del primero.

4. ¿Cuáles son los namespaces del proceso `cron`[^9]? Compare los namespaces net, ipc y uts con los del punto anterior, ¿son iguales o diferentes?

```
lrwxrwxrwx 1 root root 0 may 10 22:49 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 may 10 22:49 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 may 10 22:49 uts -> 'uts:[4026531838]'
```

Se compara con el proceso de PID 1: tienen los mismos valores de ID de namespace -> comparten el mismo namespace.

5. Usando el comando unshare crear un nuevo namespace de tipo UTS. a. unshare `--uts sh` (son dos (--) guiones juntos antes de uts) b. ¿Cuál es el nombre del host en el nuevo namespace? (comando hostname) c. Ejecutar el comando lsns. ¿Qué puede ver con respecto a los namespace?.

```
root@so:/proc# unshare --uts sh
# hostname
so
# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531834 time      133     1 root            /sbin/init
4026531835 cgroup    133     1 root            /sbin/init
4026531836 pid       133     1 root            /sbin/init
4026531837 user      133     1 root            /sbin/init
4026531838 uts       128     1 root            /sbin/init
4026531839 ipc       133     1 root            /sbin/init
4026531840 net       133     1 root            /sbin/init
4026531841 mnt       129     1 root            /sbin/init
4026532163 mnt         1   336 root            ├─/lib/systemd/systemd-udevd
4026532164 uts         1   336 root            ├─/lib/systemd/systemd-udevd
4026532165 mnt         1   399 systemd-timesync ├─/lib/systemd/systemd-timesyncd
4026532184 uts         1   399 systemd-timesync ├─/lib/systemd/systemd-timesyncd
4026532241 mnt         1   508 root            ├─/lib/systemd/systemd-logind
4026532253 uts         1   508 root            └─/lib/systemd/systemd-logind
4026531862 mnt         1    57 root            kdevtmpfs
4026532187 uts         2  5814 root            sh
```

Se muestra el nombre del host del sistema anfitrion (*so*). Además se observa que para el proceso 1 tiene un ID diferente del namespace del namespace creado anteriormente (PID 5814).

d. Modificar el nombre del host en el nuevo hostname. En el mismo entorno en el que se ejecutó `unshare` se realizó `hostname pepe-tormenta`. Al volver a revisar el nombre del hostname vemos que se actualizó. e. Abrir otra sesión, ¿cuál es el nombre del host anfitrión? Se mantiene con *so*. f. Salir del namespace (exit). ¿Qué sucedió con el nombre del host anfitrión? Se mantuvo con *so*.

Más cositas

`cgroups` ^7

Sirven para controlar la asignación de recursos a los procesos. Limitan a qué se accede y qué operaciones se permiten modificando `/etc/security/limits.conf` qeu acota la máxima asignación de recursos. Los grupos de control permiten definir jerarquías en las que se agrupan los procesos de manera que un

administrador puede definir con gran detalle la manera en la que se asignan los recursos (no solo tiempo de atención de CPU, sino también I/O y memoria principal) o llevar la contabilidad de los mismos.

Vínculo simbólico

Un vínculo simbólico (también llamado enlace simbólico o symlink) es un tipo especial de archivo en Linux que apunta a otro archivo o directorio.

- Si se borra o mueve el archivo original, el symlink queda roto.
- Se usan para abreviar rutas largas, crear alias, o gestionar versiones.

Links de interés

https://www.redhat.com/es/topics/virtualization/what-is-virtualization
https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/resource_management_guide/ch01#sec-How_Control_Groups_Are_Organized

Referencias

[^9]: demonio del sistema (es decir, un proceso que corre en segundo plano) que se encarga de ejecutar tareas programadas automáticamente en momentos específicos.