

Práctica 4A - cgroups & namespaces

Parte 1: Conceptos teóricos

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

La **virtualización** es una técnica que permite realizar una **abstracción** de los recursos físicos de una computadora, creando una capa intermedia que desacopla el hardware del sistema operativo y las aplicaciones. Gracias a esto, múltiples sistemas operativos o aplicaciones pueden ejecutarse de manera aislada sobre el mismo hardware físico, compartiendo recursos de manera eficiente y segura. **Permite:**

- Ejecutar múltiples sistemas operativos o entornos aislados sobre un mismo hardware.
- Crear "la ilusión" de tener múltiples computadoras físicas.
- Mejorar la utilización del hardware y facilitar la portabilidad, seguridad, testeo y administración de sistemas

La primera implementación significativa de virtualización fue **VM/370**, lanzado por **IBM en 1972**, para su arquitectura de mainframes **System/370**. Este sistema operativo permitía la ejecución simultánea de múltiples máquinas virtuales, cada una con su propio sistema operativo (por ejemplo, CMS - **Conversational Monitor System**), dando a cada usuario la "ilusión" de contar con una computadora completa dedicada

2. ¿Qué diferencia existe entre virtualización y emulación?

La **diferencia principal entre virtualización y emulación** radica en **cómo se interactúa con el hardware subyacente** y **cuán fiel es el entorno respecto a una máquina real o diferente**:

- **Virtualización**
 - Objetivo: Ejecutar múltiples sistemas operativos sobre el mismo tipo de hardware físico.
 - Hardware subyacente: Se usa el mismo tipo de arquitectura que la del host (por ejemplo, x86 sobre x86).
 - Performance: Alta, porque las instrucciones no privilegiadas se ejecutan directamente sobre el hardware real.
 - Requiere soporte de hardware: En muchos casos sí (por ejemplo, con KVM o Hyper-V).
- **Emulación**
 - Objetivo: Imitar completamente un sistema diferente (hardware y software), permitiendo ejecutar software compilado para una arquitectura distinta.

- Hardware subyacente: Puede ser diferente al del sistema emulado (por ejemplo, emular una SPARCstation en una PC x86).
- Performance: Baja, porque cada instrucción del sistema emulado debe interpretarse o traducirse.
- Requiere soporte de hardware: No, todo se simula por software.

3. Investigue el concepto de hypervisor y responda:

1. ¿Qué es un hypervisor?
2. ¿Qué beneficios traen los hypervisors? ¿Cómo se clasifican?

¿Qué es un hypervisor?

- Un **hypervisor** es un software especializado que **permite crear, ejecutar y gestionar máquinas virtuales (VMs)** sobre un mismo hardware físico. Su función principal es **aislar los sistemas operativos invitados (guest OS)** y permitir que compartan los recursos del hardware real (CPU, memoria, disco, etc.), controlando su acceso para evitar conflictos.

¿Qué beneficios traen los hypervisors?

- **Mejor aprovechamiento del hardware:** se pueden ejecutar múltiples sistemas sobre un solo equipo.
- **Aislamiento entre sistemas:** errores o fallos en una VM no afectan a las otras.
- **Facilidad para pruebas y desarrollo:** permite probar diferentes entornos sin comprometer el sistema base.
- **Portabilidad:** las VMs pueden moverse entre servidores fácilmente.
- **Escalabilidad y consolidación:** reduce la necesidad de servidores físicos.
- **Ahorro de energía y espacio físico:** beneficios clave en centros de datos (green IT).

¿Cómo se clasifican los hypervisors?

- **Tipo 1 – Nativo o Bare Metal**
 - Se ejecutan directamente sobre el hardware físico del host.
 - No requieren un sistema operativo anfitrión.
 - Ofrecen mejor rendimiento y menor latencia.
 - *El hypervisor corre en modo kernel (Ring 0), y los sistemas operativos invitados se ejecutan como usuarios (Ring 3).*
- **Tipo 2 – Hosted**
 - Se ejecutan como una aplicación dentro de un sistema operativo anfitrión.
 - Más fáciles de instalar y usar, ideales para pruebas y entornos personales.
 - Menor rendimiento comparado con el tipo 1 (mayor overhead).

- *Todo el hardware es emulado y las instrucciones sensibles se sustituyen por llamadas a procedimientos del hypervisor.*

4. ¿Qué es la full virtualization? ¿Y la virtualización asistida por hardware?

La **Full Virtualization** (*virtualización completa*) es una técnica de virtualización en la que se emula un entorno de hardware **completo**, permitiendo ejecutar sistemas operativos **sin necesidad de modificarlos**, como si estuvieran corriendo directamente sobre hardware físico.

Características clave:

- Los sistemas operativos invitados (guest OS) no requieren modificaciones.
- El VMM (Virtual Machine Monitor o hypervisor) intercepta las instrucciones sensibles (instrucciones privilegiadas).
- Usa técnicas como traducción binaria para reemplazar esas instrucciones por llamadas al hypervisor.
- Se ejecuta sobre la misma arquitectura que el hardware subyacente (por ejemplo, x86 sobre x86).

Ventajas:

- Alta compatibilidad: puede ejecutar cualquier SO que funcione en esa arquitectura.
- No requiere modificación del sistema operativo.

Desventajas:

- Menor rendimiento que la paravirtualización o la virtualización asistida por hardware, debido al costo de la traducción binaria.

La **virtualización asistida por hardware** es una mejora de la full virtualization que **aprovecha extensiones específicas del procesador** para gestionar directamente las instrucciones sensibles, **sin necesidad de traducirlas** o emularlas por software. **Características clave:**

- Requiere CPU con soporte de virtualización, como:
 - Intel VT-x (Virtualization Technology)
 - AMD-V (Secure Virtual Machine)
- Introduce modos especiales de ejecución:
 - Intel: Root Mode (para el VMM) y Non-Root Mode (para las VMs).
- Tanto el hypervisor como el guest OS se ejecutan en Ring 0, pero en modos separados.
- Las instrucciones privilegiadas ejecutadas por el guest generan traps automáticamente al hypervisor.

Ventajas:

- Mejor rendimiento que la full virtualization con traducción binaria.
- Permite virtualizar sin modificar los sistemas operativos, como en full virtualization.

- Soporte más eficiente para múltiples VMs concurrentes.

Requiere:

- Un procesador moderno con soporte de virtualización activado en la BIOS/UEFI.

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

La **Binary Translation** (traducción binaria) es una técnica en la cual el **hypervisor analiza dinámicamente el código del sistema operativo invitado** y **reemplaza las instrucciones sensibles o problemáticas** (que podrían comprometer el control del hypervisor) por llamadas seguras al mismo. **Características:**

- Funciona en tiempo de ejecución: el hypervisor escanea bloques de código ("basic blocks") antes de ejecutarlos.
- Los bloques que contienen instrucciones sensibles son modificados y luego almacenados en caché para mejorar el rendimiento en futuras ejecuciones.
- Los bloques sin instrucciones sensibles se ejecutan directamente en el hardware.

Ventajas:

- Permite virtualizar sin modificar el sistema operativo invitado.
- Compatible con arquitecturas no diseñadas originalmente para virtualización (como x86).

Desventajas:

- Tiene overhead de rendimiento, especialmente si hay muchas instrucciones sensibles.
- Mayor complejidad en la implementación.

Trap-and-Emulate es una técnica clásica en la virtualización en la que el **guest OS se ejecuta en modo usuario**, y **cuando intenta ejecutar una instrucción privilegiada**, esta genera una **interrupción o trap** que es capturada por el hypervisor, quien **emula** el comportamiento deseado de esa instrucción. **Características:**

- El VMM (hypervisor) se ejecuta en modo privilegiado (Ring 0).
- El sistema operativo invitado se ejecuta en modo usuario (Ring 3).
- Las instrucciones privilegiadas no se ejecutan directamente, sino que activan una excepción.
- El hypervisor interpreta o emula la instrucción y luego devuelve el control al guest.

Ventajas:

- Técnica simple y directa cuando el hardware lo permite.
- Utilizada en arquitecturas donde las instrucciones sensibles también son privilegiadas (condición del teorema de Popek y Goldberg).

Desventajas:

- No aplicable directamente en x86 clásico, ya que muchas instrucciones sensibles no son privilegiadas y por lo tanto no generan traps. Esto motivó el uso de traducción binaria.

6. Investigue el concepto de paravirtualización y responda:

1. ¿Qué es la paravirtualización?
2. Mencione algún sistema que implemente paravirtualización.
3. ¿Qué beneficios trae con respecto al resto de los modos de virtualización?

¿Qué es la paravirtualización?

- La **paravirtualización** es una técnica de virtualización en la que el sistema operativo invitado (**guest OS**) está **modificado intencionalmente** para poder interactuar de manera más eficiente con el **hypervisor**. En lugar de ejecutar directamente instrucciones sensibles o privilegiadas del procesador (que podrían fallar o no ser atrapadas), el guest realiza **llamadas explícitas** al hypervisor a través de una **API especial** llamadas **hypercalls**.

Características clave:

- El sistema operativo guest es consciente de que está virtualizado.
- Se evita la emulación o traducción binaria de instrucciones sensibles.
- Las instrucciones sensibles son reemplazadas por llamadas directas al hypervisor

Ejemplos de sistemas que la implementan

- **Xen Project**: uno de los pioneros en paravirtualización. Permite ejecutar tanto sistemas modificados como no modificados (con soporte asistido por hardware).
- **KVM**: permite paravirtualización parcial a través de **drivers especiales**, como VirtIO.
- **VMware y Hyper-V**: también pueden usar **drivers paravirtualizados** (por ejemplo, para dispositivos de red o almacenamiento) dentro de VMs que usan virtualización completa.

¿Qué beneficios trae con respecto al resto de los modos de virtualización?

Beneficio	Explicación
Mejor rendimiento	Al evitar la traducción binaria y emulación de hardware, se reduce el overhead.
Menor latencia	Las llamadas al hypervisor son más directas (via API).
Menor complejidad en el VMM	El hypervisor no necesita traducir instrucciones sensibles.
Más control sobre recursos	El guest puede cooperar con el hypervisor para optimizar el uso de CPU, memoria, etc.

Desventajas:

- Requiere modificar el sistema operativo guest, por lo que no todos los sistemas operativos son compatibles.
- Menor portabilidad en comparación con la virtualización completa.

7. Investigue sobre containers y responda:

1. ¿Qué son?
2. ¿Dependen del hardware subyacente?
3. ¿Qué lo diferencia por sobre el resto de las tecnologías estudiadas?
4. Investigue qué funcionalidades son necesarias para poder implementar containers.

¿Qué son los containers?

- Los **containers** (contenedores) son una forma de **virtualización ligera a nivel de sistema operativo**, que permite ejecutar múltiples entornos aislados sobre el mismo kernel del sistema operativo anfitrión. Cada contenedor contiene **una aplicación o conjunto de procesos** con sus propias dependencias, configuraciones y sistema de archivos, pero todos comparten el **mismo kernel del host**.

¿Dependen del hardware subyacente?

- **No directamente.** Los contenedores **no virtualizan el hardware** como hacen los hypervisores. En cambio, se ejecutan como **procesos normales del sistema operativo anfitrión**, lo que significa que no requieren virtualización asistida por hardware ni emulación.

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

Característica	Containers	Máquinas Virtuales (Hypervisor)
Nivel de virtualización	A nivel de sistema operativo	A nivel de hardware
Kernel	Compartido entre host y containers	Independiente para cada VM
Rendimiento	Muy alto (casi nativo)	Más bajo (por emulación/aislamiento)
Uso de recursos	Bajo (ligero)	Alto (requiere más RAM/CPU)
Portabilidad	Alta	Media
Tiempo de arranque	Segundos	Minutos
Soporte de otros SO	Solo el mismo kernel	Cualquier SO compatible

¿Qué funcionalidades del sistema son necesarias para implementar containers?

- **Se necesitan estas 2 funcionalidades:**
 - **Namespaces**

- Proveen aislamiento de recursos.
- *cgroups (Control Groups)*
 - Permiten limitar, controlar y monitorear el uso de recursos por parte de un contenedor.

Parte 2: chroot, Control Groups y Namespaces

Nota >

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

Nota >

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 `chroot` (change root) es una utilidad del sistema Unix/Linux que permite **cambiar el directorio raíz aparente (/)** para un proceso determinado y sus procesos hijos. A partir de ese momento, ese proceso "verá" el nuevo directorio especificado como si fuera el **sistema de archivos completo**.

La finalidad de `chroot` es crear un entorno **aislado** o **contenedor básico**, comúnmente conocido como **"chroot jail" (jaula chroot)**, donde un proceso:

- No puede acceder a archivos fuera de ese nuevo entorno raíz.
- Cree que está ejecutándose en un sistema aislado.
- Tiene acceso solo a los recursos y comandos disponibles dentro de esa raíz modificada

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?

Salida del comando:

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

Vemos esa salida porque el entorno `chroot` está incompleto. En un sistema `chroot`, el directorio raíz (/) se cambia a la ubicación que especifiquemos, pero el entorno dentro de esa ubicación necesita contener los archivos y bibliotecas necesarias para ejecutar comandos como `/bin/bash`.

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?

Salida del comando:

```
root@so:~/sobash# ldd /bin/bash
        linux-vdso.so.1 (0x00007f4344749000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6
(0x00007f43445ca000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f43443e9000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f434474b000)
```

El archivo `linux-vdso.so.1` que aparece en la salida de `ldd /bin/bash` no se encuentra en un directorio físico dentro del sistema de archivos. `linux-vdso.so.1` es un archivo virtual que el kernel de Linux utiliza para proporcionar funciones de espacio de usuario que se encuentran en el espacio de direcciones del proceso de manera eficiente. **VDSO** significa **Virtual Dynamic Shared Object**. No es un archivo real en el sistema de archivos, sino que es proporcionado dinámicamente por el kernel para acelerar ciertas llamadas al sistema. Debido a que `linux-vdso.so.1` es un objeto compartido virtual generado por el propio núcleo del sistema operativo, no se almacena en el sistema de archivos como un archivo regular. Esto significa que **no necesitamos copiarlo ni gestionarlo manualmente dentro de nuestro entorno `chroot`**; es proporcionado directamente por el kernel cuando se ejecuta el programa. Es uno de los mecanismos internos del sistema para mejorar el rendimiento de ciertas operaciones de la API de Linux.

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?

Comandos que hacemos:

```
root@so:~# cp /bin/bash /root/sobash/bin/
root@so:~# ldd /bin/bash
        linux-vdso.so.1 (0x00007fccb163c000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6
(0x00007fccb14bd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fccb12dc000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fccb163e000)
root@so:~# cp /lib/x86_64-linux-gnu/libtinfo.so.6 /root/sobash/lib/x86_64-
linux-gnu/
root@so:~# cp /lib/x86_64-linux-gnu/libc.so.6 /root/sobash/lib/x86_64-linux-
gnu/
root@so:~# cp /lib64/ld-linux-x86-64.so.2 /root/sobash/lib64/
root@so:~# chroot /root/sobash
bash-5.2#
```

Al ver el prompt `bash-5.2#`, significa que estamos dentro del entorno `chroot`, corriendo una shell funcional desde el entorno que creamos en `/root/sobash`.

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

Dentro del entorno puedo ejecutar los comandos `cd` y `echo`, esto se debe a que son comandos **builtin** de Bash, no dependen de ningún archivo externo. `ls` no lo pude ejecutar ya que necesitamos copiar manualmente el binario de `ls` y sus dependencias al entorno.

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

Al ejecutar `pwd` nosotros vemos `/`, esto se debe a que **dentro del entorno** `chroot`, el directorio raíz `/` **ya no es el del sistema original**, sino el del nuevo entorno que definimos (en este caso, `/root/sobash`).

8. Salir del entorno `chroot` usando `exit`.
9. Usando el repositorio de la cátedra acceda a los materiales en practica4/02-chroot:
 1. Verifique que tiene instalado `busybox` en `/bin/busybox`
 2. Cree un `chroot` con `busybox` usando `/buildbusyboxroot.sh`
 3. Entre en el `chroot`
 4. Busque el directorio `/home/so` ¿Qué sucede? ¿Por qué?

5. Ejecute el comando "ps aux" ¿Qué procesos ve? ¿Por qué (pista: ver el contenido de /proc)?
6. Monte /proc con "mount -t proc proc /proc" y vuelva a ejecutar "ps aux" ¿Qué procesos ve? ¿Por qué?
7. Acceda a /proc/1/root/home/so ¿Qué sucede?
8. ¿Qué conclusiones puede sacar sobre el nivel de aislamiento provisto por chroot?

Verificamos que tenemos instalado busybox

```
so@so:~/codigo-para-practicas/practica4/02-chroot$ ls -l /bin/busybox
-rwxr-xr-x 1 root root 772880 abr 23 2023 /bin/busybox
```

Cambios que tuve que hacer en el código para que me funcione

```
Línea 39 original: chroot busyboxroot /bin/busybox --install -s /bin
Línea 39 nueva: /usr/sbin/chroot busyboxroot /bin/busybox --install -s /bin
```

Creamos el entorno y entramos al mismo

```
so@so:~/codigo-para-practicas/practica4/02-chroot$ su -c
"./buildbusyboxroot.sh"
Contraseña:
    linux-vdso.so.1 (0x00007f174418e000)
    libresolv.so.2 ⇒ /lib/x86_64-linux-gnu/libresolv.so.2
(0x00007f17440b3000)
    libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1743ed2000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f1744190000)
BusyBox root filesystem created in /home/so/codigo-para-
practicas/practica4/02-chroot/busyboxroot
You can now chroot into it with:
chroot /home/so/codigo-para-practicas/practica4/02-chroot/busyboxroot
/bin/sh
so@so:~/codigo-para-practicas/practica4/02-chroot$ su -c "/usr/sbin/chroot
/home/so/codigo-para-practicas/practica4/02-chroot/busyboxroot /bin/sh"
Contraseña:

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

/ #
```

Buscamos /home/so

```
/ # ls /home/so
ls: /home/so: No such file or directory
```

No lo encuentra porque `chroot` creó un entorno aislado que parte desde `busyboxroot/` como su raíz, por lo tanto, la carpeta que buscamos no existe ahí al menos que la hayamos copiado previamente

Salida del comando `ps aux`

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

No aparece ningún proceso listado, esto ocurre porque:

- Dentro del `chroot`, **no hay acceso a `/proc`**, y `ps` obtiene la información de los procesos leyendo los archivos de `/proc`.
- Como no está montado `/proc`, `ps` "cree" que no hay procesos corriendo.

Procesos después de montar `/proc`

```
/ # 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_]
6 0      [kworker/R-slub_]
7 0      [kworker/R-netns]
9 0      [kworker/0:1-cgr]
10 0     [kworker/0:0H-ev]
11 0     [kworker/u16:0-i]
12 0     [kworker/R-mm_pe]
13 0     [rcu_tasks_kthre]
14 0     [rcu_tasks_rude_]
15 0     [rcu_tasks_trace]
16 0     [ksoftirqd/0]
17 0     [rcu_preempt]
18 0     [rcu_exp_par_gp_]
19 0     [rcu_exp_gp_kthr]
20 0     [migration/0]
```

```
21 0      [idle_inject/0]
22 0      [cpuhp/0]
23 0      [cpuhp/1]
24 0      [idle_inject/1]
25 0      [migration/1]
26 0      [ksoftirqd/1]
28 0      [kworker/1:0H-kb]
29 0      [cpuhp/2]
30 0      [idle_inject/2]
31 0      [migration/2]
32 0      [ksoftirqd/2]
35 0      [cpuhp/3]
36 0      [idle_inject/3]
37 0      [migration/3]
38 0      [ksoftirqd/3]
40 0      [kworker/3:0H-kb]
41 0      [kworker/u17:0-f]
42 0      [kworker/u18:0-f]
44 0      [kworker/u20:0-e]
47 0      [kworker/u18:2-e]
50 0      [kworker/u17:2-e]
52 0      [kdevtmpfs]
53 0      [kworker/R-inet_]
54 0      [kauditd]
55 0      [oom_reaper]
56 0      [kworker/R-write]
57 0      [kcompactd0]
58 0      [ksmd]
59 0      [khugepaged]
60 0      [kworker/R-kinte]
61 0      [kworker/R-kbloc]
62 0      [kworker/R-blkcq]
63 0      [irq/9-acpi]
65 0      [kworker/R-tpm_d]
66 0      [kworker/R-edac-]
67 0      [kworker/R-devfr]
68 0      [kworker/2:1-mm_]
70 0      [kswapd0]
73 0      [kworker/u19:1-e]
79 0      [kworker/R-kthro]
84 0      [kworker/R-acpi_]
85 0      [kworker/R-ml_d]
87 0      [kworker/R-ipv6_]
93 0      [kworker/R-kstrp]
97 0      [kworker/u21:0]
98 0      [kworker/u22:0]
```

```

99 0      [kworker/u23:0]
100 0     [kworker/u24:0]
101 0     [kworker/u25:0]
173 0     [kworker/0:1H-kb]
207 0     [kworker/3:1H]
219 0     [kworker/2:1H-kb]
223 0     [kworker/3:2-eve]
230 0     [kworker/R-ata_s]
231 0     [kworker/2:2-cgr]
232 0     [scsi_eh_0]
233 0     [kworker/R-scsi_]
234 0     [scsi_eh_1]
235 0     [kworker/R-scsi_]
236 0     [kworker/u20:2-e]
237 0     [scsi_eh_2]
238 0     [kworker/R-scsi_]
240 0     [kworker/0:2-eve]
272 0     [jbd2/sda1-8]
273 0     [kworker/R-ext4-]
313 0     /lib/systemd/systemd-journald
344 0     /lib/systemd/systemd-udev
415 997   /lib/systemd/systemd-timesyncd
429 0     [kworker/R-crypt]
481 0     dhclient -4 -v -i -pf /run/dhclient.enp0s8.pid -lf
/var/lib/dhcp/dhclient.enp0s8
487 0     dhclient -4 -v -i -pf /run/dhclient.enp0s3.pid -lf
/var/lib/dhcp/dhclient.enp0s3
530 0     /usr/sbin/cron -f
531 100   /usr/bin/dbus-daemon --system --address=systemd: --nofork --
nopicfile --systemd-
535 0     /lib/systemd/systemd-logind
555 0     /sbin/wpa_supplicant -u -s -O DIR=/run/wpa_supplicant
GROUP=netdev
576 0     [irq/18-vmwgfx]
580 0     [kworker/R-ttm]
599 0     [kworker/u16:2-i]
632 0     /bin/login -p --
643 0     sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
681 0     [kworker/u19:3-e]
682 0     [kworker/u19:4-e]
688 0     [kworker/2:0H-kb]
693 0     [kworker/u17:1-e]
699 1000  /lib/systemd/systemd --user
700 1000  (sd-pam)
716 1000  -bash
719 0     sshd: so [priv]

```

```

725 1000      sshd: so@notty
726 1000      sh
744 1000      /home/so/.vscode-server/code-
19e0f9e681ecb8e5c09d8784acaa601316ca4571 command-sh
797 1000      sh /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316
801 1000      /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
832 1000      /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
844 1000      /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
887 1000      /bin/bash --init-file /home/so/.vscode-
server/cli/servers/Stable-19e0f9e681ecb8e
938 1000      /bin/sh
1044 0        [kworker/1:1H-kb]
1241 0        [kworker/u20:1-e]
1253 0        [kworker/3:1-cgr]
1340 0        [kworker/1:1-eve]
1558 0        [kworker/u18:1-e]
1847 0        [kworker/1:0-ata]
1953 0        su -c /usr/sbin/chroot /home/so/codigo-para-
practicas/practica4/02-chroot/busybo
1957 0        /bin/sh
2075 0        [kworker/1:2-ata]
2076 0        [kworker/u18:3-e]
2077 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
2158 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
2164 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4
2177 0        [kworker/2:2H-kb]
2217 1000     sleep 180
2259 1000     /bin/sh -c "/home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784a
2260 1000     {cpuUsage.sh} /bin/bash /home/so/.vscode-
server/cli/servers/Stable-19e0f9e681ecb
2265 1000     sleep 1
2266 0        ps aux

```

Una vez montado `/proc`, `ps aux` muestra:

- El proceso init o `systemd` (PID 1).
- Todos los *kernel threads* (`[kworker/ ...]`, `[rcu/ ...]`, etc.).

- Procesos de usuario como `bash`, `sshd`, `dhclient`, `cron`, `dbus-daemon`.
- Mi shell dentro del entorno `chroot`.
- Procesos de VS Code Server (`.vscode-server/cli/servers/...`) porque estoy accediendo desde VS Code con SSH.

Vemos esos procesos porque el comando `ps` utiliza `/proc` - para obtener la lista de procesos y detalles como PID, uso de CPU, memoria, etc. Como antes no estaba montado `/proc` no podíamos ver ningún proceso.

Acceso a `/proc/1/root/home/so`

```
/ # ls /proc/1/root/home/so
codigo-para-practicas  kernel                practica2
install_deps.sh        practica1
```

Vemos esto porque **el proceso 1 (init) no está chrootado** y tiene acceso completo al sistema de archivos real. Al acceder a `/proc/1/root/home/so`, **estamos viendo el contenido de `/home/so` desde la raíz real del sistema**, sin importar si nuestra shell actual está dentro de un entorno `chroot`.

Conclusiones

El mecanismo `chroot` proporciona un aislamiento básico a nivel de sistema de archivos, pero **no constituye una medida de seguridad fuerte**.

- No aísla procesos, red ni usuarios: solo cambia la raíz del sistema de archivos visible para el proceso.
- Es posible escapar del entorno `chroot` (por ejemplo, accediendo vía `/proc/<pid>/root`) si el proceso tiene privilegios elevados.
- Otros procesos del sistema que no están en el `chroot` pueden acceder al entorno `chrootado`.
- **No previene el acceso indirecto** a archivos fuera del entorno mediante el uso del sistema de archivos `/proc`.

Por todo esto, `chroot` **no debe utilizarse como mecanismo de seguridad**, sino únicamente como una herramienta práctica para pruebas, entornos controlados o recuperación de sistemas

Control Groups

 **Nota** >

Se aconseja realizar esta parte de la práctica en una máquina virtual (por ejemplo en la provista por la práctica) ya que es necesario cambiar la configuración de CGroups.

Preparación

Nota >

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"
```

Realizamos la edición (yo lo hice con nano)

```
so@so:~/codigo-para-practicas/practica4/02-chroot$ su -c "nano
/etc/default/grub"
Contraseña:
so@so:~/codigo-para-practicas/practica4/02-chroot$ cat /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet systemd.unified_cgroup_hierarchy=0"
GRUB_CMDLINE_LINUX=""

# If your computer has multiple operating systems installed, then you
# probably want to run os-prober. However, if your computer is a host
# for guest OSes installed via LVM or raw disk devices, running
# os-prober can cause damage to those guest OSes as it mounts
# filesystems to look for things.
#GRUB_DISABLE_OS_PROBER=false
```



```
# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to
Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
```

2. Actualizar la configuración de GRUB: `sudo update-grub`

```
so@so:~/codigo-para-practicas/practica4/02-chroot$ sudo update-grub
[sudo] contraseña para so:
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.13.7
Found initrd image: /boot/initrd.img-6.13.7
Found linux image: /boot/vmlinuz-6.13.7.old
Found initrd image: /boot/initrd.img-6.13.7
Found linux image: /boot/vmlinuz-6.1.0-31-amd64
Found initrd image: /boot/initrd.img-6.1.0-31-amd64
Found linux image: /boot/vmlinuz-6.1.0-29-amd64
Found initrd image: /boot/initrd.img-6.1.0-29-amd64
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
```

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:/$ 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
```

Nota >

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`.

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

Los **cgroups** están montados en el sistema de archivos especial ubicado en: `/sys/fs/cgroup/`. Cuando hacemos el `ls` se ven los directorios: `blkio cpu cpuacct devices freezer memory pids systemd` etc. estos son **controladores de Cgroups v1** montados por separado, lo cual indica que el sistema está utilizando **Cgroups versión 1 (v1)**. Además, si vemos un directorio llamado `unified`, eso indica la presencia de **Cgroups v2**, aunque no necesariamente esté en uso activo.

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

Para verificar esto podemos hacer: `cat /sys/fs/cgroup/unified/cgroup.controllers`. Este archivo lista los controladores que soporta cgroups v2, en mi caso veo: `cpuset memory`. Por lo tanto, si ese archivo existe, entonces el kernel tiene **cgroups v2 habilitado**, aunque no esté usándose como la jerarquía principal.

7. Analice qué sucede si se remueve un controlador de cgroups v1 (por ej. `Umount /sys/fs/cgroup/rdma`).

```
so@so:/$ su -c "umount /sys/fs/cgroup/rdma"
Contraseña:
```

Acá lo pudimos desmontar porque **no está en uso por ningún proceso ni tiene cgroups activos**, sino, no podríamos desmontarlo. Al desmontarlo **no vamos a poder crear nuevos**

cgroups bajo ese controlador, las herramientas que usan cgroups para controlar el recurso asociado (en este caso, RDMA) no podrán aplicarlo hasta que se remonte y aunque no se rompe el sistema, sí puede afectar scripts o servicios que esperen que ese controlador esté montado.

8. Crear dos cgroups dentro del subsistema cpu llamados cpualta y cpubaja.

```
so@so:/$ su -c "mkdir /sys/fs/cgroup/cpu/cpualta"
Contraseña:
so@so:/$ su -c "mkdir /sys/fs/cgroup/cpu/cpubaja"
Contraseña:
so@so:/$ ls /sys/fs/cgroup/cpu/
cgroup.clone_children      cpuacct.usage_sys      cpu.stat
sys-kernel-config.mount
cgroup.procs               cpuacct.usage_user     cpu.stat.local
sys-kernel-debug.mount
cgroup.sane_behavior       cpualta                dev-hugepages.mount
sys-kernel-tracing.mount
cpuacct.stat               cpubaja                dev-mqueue.mount
system.slice
cpuacct.usage              cpu.cfs_burst_us       init.scope
tasks
cpuacct.usage_all          cpu.cfs_period_us      notify_on_release
user.slice
cpuacct.usage_percpu       cpu.cfs_quota_us       proc-sys-fs-binfmt_misc.mount
cpuacct.usage_percpu_sys   cpu.idle                release_agent
cpuacct.usage_percpu_user   cpu.shares              sys-fs-fuse-connections.mount
```

9. Controlar que se hayan creado tales directorios y ver si tienen algún contenido `# mkdir /sys/fs/cgroup/cpu/"nombre_cgroup"`

```
so@so:/$ ls /sys/fs/cgroup/cpu/cpualta
cgroup.clone_children  cpuacct.usage_all      cpuacct.usage_sys
cpu.cfs_quota_us      cpu.stat.local
cgroup.procs          cpuacct.usage_percpu   cpuacct.usage_user
cpu.idle               notify_on_release
cpuacct.stat           cpuacct.usage_percpu_sys  cpu.cfs_burst_us
cpu.shares             tasks
cpuacct.usage          cpuacct.usage_percpu_user  cpu.cfs_period_us
cpu.stat
so@so:/$ ls /sys/fs/cgroup/cpu/cpubaja
cgroup.clone_children  cpuacct.usage_all      cpuacct.usage_sys
cpu.cfs_quota_us      cpu.stat.local
cgroup.procs          cpuacct.usage_percpu   cpuacct.usage_user
```

```

cpu.idle          notify_on_release
cpuacct.stat      cpuacct.usage_percpu_sys  cpu.cfs_burst_us
cpu.shares        tasks
cpuacct.usage     cpuacct.usage_percpu_user  cpu.cfs_period_us
cpu.stat

```

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

Se está utilizando **cgroups v1**, porque:

- Existen múltiples subdirectorios bajo `/sys/fs/cgroup/` (como `cpu/`, `memory/`, `blkio/`, etc.), lo cual es característico de **cgroups v1**, donde cada controlador tiene su propio punto de montaje.
 - Dentro de `/sys/fs/cgroup/cpu/`, los grupos `cpualta` y `cpubaja` contienen archivos como `cpu.shares`, `cpu.cfs_quota_us`, y `tasks`, todos propios de la interfaz **v1 del controlador CPU**.
11. 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

```

12. 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`.
13. 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 &`

En termalta

```

root@so:~# taskset -c 0 md5sum /dev/urandom &
[1] 1718

```

En termbaja

```

root@so:~# taskset -c 0 md5sum /dev/urandom &
[1] 1740

```

14. 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?

Parte de la salida del top

```
top - 11:08:15 up 10 min, 1 user, load average: 10,24, 7,55, 3,91
Tareas: 139 total, 5 running, 134 sleeping, 0 stopped, 0 zombie
%Cpu(s): 13,2 us, 83,8 sy, 0,0 ni, 0,9 id, 0,0 wa, 0,0 hi, 2,1 si, 0,0 st
MiB Mem : 8879,5 total, 6948,3 free, 1659,7 used, 546,5 buff/cache
MiB Intercambio: 975,0 total, 975,0 free, 0,0 used. 7219,8
avail Mem
```

	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
COMMAND											
	1328	so	20	0	45820	27448	3240	S	149,1	0,3	11:52.21 rg
	1004	so	20	0	149060	39224	3240	S	143,7	0,4	18:35.00 rg
	1718	root	20	0	5476	1840	1712	R	43,4	0,0	1:50.86
md5sum											
	1740	root	20	0	5476	1768	1640	R	38,3	0,0	1:48.91
md5sum											

Los 2 procesos aproximadamente se manejan entre un 30% y un 50% de la CPU

15. 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

En termalta

```
root@so:~# echo 1718 > /sys/fs/cgroup/cpu/cpualta/cgroup.procs
root@so:~# cat /sys/fs/cgroup/cpu/cpualta/cgroup.procs
1718
```

En termbaja

```
root@so:~# echo 1740 > /sys/fs/cgroup/cpu/cpubaja/cgroup.procs
root@so:~# cat /sys/fs/cgroup/cpu/cpubaja/cgroup.procs
1740
```

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

Parte de la salida del `top`

```
top - 11:21:44 up 23 min, 1 user, load average: 9,43, 8,73, 6,73
Tareas: 138 total, 4 running, 134 sleeping, 0 stopped, 0 zombie
%Cpu(s): 13,6 us, 81,5 sy, 0,0 ni, 2,7 id, 0,0 wa, 0,0 hi, 2,2 si, 0,0 st
MiB Mem : 8879,5 total, 6063,4 free, 2543,6 used, 547,5 buff/cache
MiB Intercambio: 975,0 total, 975,0 free, 0,0 used. 6335,9
avail Mem
```

	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
COMMAND											
	1004	so	20	0	149196	38936	3240	S	147,6	0,4	35:48.82 rg
	1328	so	20	0	45952	28496	3240	S	133,3	0,3	33:20.81 rg
	1718	root	20	0	5476	1840	1712	R	70,2	0,0	8:31.12
md5sum											
	1740	root	20	0	5476	1768	1640	R	30,1	0,0	6:56.48
md5sum											

Ahora el proceso de termalta recibe un 70% de la cpu y el de termbaja un 30%.

17. 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?

kill

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

Parte de la salida del `top`

```
top - 11:24:20 up 26 min, 1 user, load average: 9,16, 8,93, 7,12
Tareas: 137 total, 2 running, 135 sleeping, 0 stopped, 0 zombie
%Cpu(s): 17,5 us, 77,0 sy, 0,0 ni, 3,2 id, 0,0 wa, 0,0 hi, 2,3 si, 0,0 st
MiB Mem : 8879,5 total, 5889,9 free, 2717,1 used, 547,7 buff/cache
MiB Intercambio: 975,0 total, 975,0 free, 0,0 used. 6162,4
avail Mem
```

	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
COMMAND											
	1328	so	20	0	45972	28624	3240	S	144,9	0,3	37:24.53 rg
	1004	so	20	0	149224	38384	3240	S	116,7	0,4	39:03.39 rg

```
1740 root      20    0   5476   1768   1640 R   91,8    0,0    8:23.59
md5sum
```

Como solo el proceso de `termbaja` ahora está ejecutando, no hace falta que respete su restricción, lo que le permite tomar mas porcentaje de la CPU.

18. Finalizar el otro proceso `md5sum`.

19. 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)
```

20. 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?

Análisis del `top`

```
top - 11:42:12 up 44 min,  1 user,  load average: 9,71, 9,33, 8,50
Tareas:  4 total,  2 running,  2 sleeping,  0 stopped,  0 zombie
%Cpu(s): 15,1 us, 76,4 sy,  0,0 ni,  3,1 id,  0,0 wa,  0,0 hi,  5,4 si,  0,0 st
MiB Mem :  8879,5 total,  4643,1 free,  3920,1 used,   591,0 buff/cache
MiB Intercambio:   975,0 total,   975,0 free,    0,0 used.  4959,4
avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
7261	root	20	0	5476	1876	1748	R	70,5	0,0	10:33.07
md5sum										
7311	root	20	0	5476	1960	1832	R	29,8	0,0	4:29.22
md5sum										
1555	root	20	0	10596	6412	4236	S	0,0	0,1	0:00.07
1623	root	20	0	9056	5700	3652	S	0,0	0,1	0:00.03
										bash
										bash

Se sigue manteniendo la restricción de la cpu dependiendo el cgroup, lo que ocurre con el `top` y los procesos de los shells es que `top` (por defecto) **ordena por uso de CPU** y muestra los procesos más activos. Los shells (`bash`) están prácticamente inactivos mientras esperan que escribamos algo, por eso:

- Sí están corriendo.

- Pero consumen **0% de CPU**, así que no aparecen arriba en `top`.

21. 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?

Nueva salida del `top`

```
top - 11:50:12 up 52 min, 1 user, load average: 9,25, 8,98, 8,60
Tareas: 5 total, 3 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 14,5 us, 81,9 sy, 0,0 ni, 2,2 id, 0,0 wa, 0,0 hi, 1,3 si, 0,0 st
MiB Mem : 8879,5 total, 4134,5 free, 4428,4 used, 591,7 buff/cache
MiB Intercambio: 975,0 total, 975,0 free, 0,0 used. 4451,1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
7261	root	20	0	5476	1876	1748	R	70,6	0,0	16:08.47
7311	root	20	0	5476	1960	1832	R	17,8	0,0	6:33.67
11416	root	20	0	5476	1752	1624	R	11,7	0,0	0:19.51
1555	root	20	0	10596	6412	4236	S	0,0	0,1	0:00.08 bash
1623	root	20	0	9056	5700	3652	S	0,0	0,1	0:00.04 bash

Al tener ahora **3 procesos `md5sum` ejecutándose simultáneamente** (1 en `cpualta`, 2 en `cpubaja`) lo que pasa es que:

- El proceso `7261` (de `cpualta`) está usando ~70% del CPU.
- Los dos procesos de `cpubaja` (`7311` y `11416`) **se reparten el 30% restante**, respetando la limitación total del cgroup.

La razón es que los **cgroups de CPU controlan la proporción de CPU disponible** para los procesos dentro de ellos. En este caso:

- `cpualta` fue configurado con **70% de uso permitido**.
- `cpubaja` tiene **30% de uso permitido**.

Esto quiere decir:

- Todos los procesos dentro del cgroup `cpualta` **comparten un 70% del CPU**.
- Todos los procesos dentro del cgroup `cpubaja` **comparten un 30% del CPU**.

Namespaces

1. Explique el concepto de namespaces.

Los **namespaces** son una característica del kernel de Linux que permite **aislar recursos del sistema** para que un conjunto de procesos vea una **versión limitada o personalizada** de esos recursos. En otras palabras, cada namespace crea una "vista" parcial del sistema, lo que permite que **procesos distintos creen estar en entornos separados**, aunque estén en el mismo sistema operativo.

2. ¿Cuáles son los posibles namespaces disponibles?

Tipo de Namespace	Recurso que aísla
PID	Árbol de procesos (cada uno con su PID 1)
Mount	Sistema de archivos y puntos de montaje
Network	Interfaces de red, IPs, puertos, etc.
UTS	Nombre del host y dominio
IPC	Colas de mensajes, semáforos, memoria compartida
User	IDs de usuario y grupo
Cgroup	Vista y jerarquía de control groups
Time	Tiempo del sistema (offset por namespace)

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)?

```
root@so:~# ls -l /proc/1/ns/{net,ipc,uts}
lrwxrwxrwx 1 root root 0 may 14 11:56 /proc/1/ns/ipc → 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 may 14 11:56 /proc/1/ns/net → 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 may 14 11:56 /proc/1/ns/uts → 'uts:[4026531838]'
```

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

```
root@so:~# pidof cron
602
root@so:~# ls -l /proc/602/ns/{net,ipc,uts}
lrwxrwxrwx 1 root root 0 may 14 11:58 /proc/602/ns/ipc → 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 may 14 11:58 /proc/602/ns/net → 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 may 14 11:58 /proc/602/ns/uts → 'uts:[4026531838]'
```

Son **iguales**, por lo tanto, `cron` es un proceso que **comparte los mismos namespaces del sistema base**, y no está aislado.

5. Usando el comando `unshare` crear un nuevo namespace de tipo UTS.

1. `unshare -uts sh` (son dos `-` guiones juntos antes de `uts`)
2. ¿Cuál es el nombre del host en el nuevo namespace? (comando `hostname`)
3. Ejecutar el comando `lsns`. ¿Qué puede ver con respecto a los namespace?.
4. Modificar el nombre del host en el nuevo `hostname`.
5. Abrir otra sesión, ¿cuál es el nombre del host anfitrión?
6. Salir del namespace (`exit`). ¿Qué sucedió con el nombre del host anfitrión?

Creación del namespace y nombre de host

```
root@so:~# unshare --uts sh
# hostname
so
```

El **hostname es igual al del sistema anfitrión**, porque al crear un namespace UTS nuevo, se copia el valor actual del `hostname`, pero se vuelve **aislado**: podemos cambiarlo dentro del namespace sin que afecte al resto del sistema.

Comando `lsns`

```
# lsns
```

	NS	TYPE	NPROCS	PID	USER	COMMAND
4026531834	time		135	1	root	/sbin/init
4026531835	cgroup		135	1	root	/sbin/init
4026531836	pid		135	1	root	/sbin/init
4026531837	user		135	1	root	/sbin/init
4026531838	uts		130	1	root	/sbin/init
4026531839	ipc		135	1	root	/sbin/init
4026531840	net		135	1	root	/sbin/init
4026531841	mnt		131	1	root	/sbin/init
4026532163	mnt		1	345	root	/lib/systemd/systemd-udev
4026532164	uts		1	345	root	/lib/systemd/systemd-udev
4026532165	mnt		1	382	systemd-timesync	/lib/systemd/systemd-
	timesyncd					timesyncd
4026532239	uts		1	382	systemd-timesync	/lib/systemd/systemd-
	timesyncd					timesyncd
4026532294	mnt		1	608	root	/lib/systemd/systemd-
	logind					logind
4026532296	uts		1	608	root	/lib/systemd/systemd-
	logind					logind

```
4026531862 mnt      1    49 root      kdevtmpfs
4026532240 uts      2 13052 root      sh
```

4026532240 uts 2 13052 root sh indica que hay un **nuevo namespace UTS (con ID 4026532240)** y que **el proceso sh (PID 13052)** lo está utilizando. `lsns` nos permite ver todos los namespaces activos del sistema, agrupados por tipo (`uts` , `net` , etc.) y muestra cuántos procesos están asociados a cada uno.

Modificando el nombre de host

```
# hostname nuevohost
# hostname
nuevohost
```

Esto **solo afecta el hostname dentro del namespace**. El sistema principal no lo ve. En otra consola podemos ver que el `hostname` será el del **sistema anfitrión**, en nuestro caso `so` . Esto demuestra que el **host original sigue teniendo su propio nombre**, porque el cambio hecho dentro del namespace es aislado.

Saliendo del namespace

```
# exit
root@so:~# hostname
so
```

El cambio del hostname en el namespace **no afectó al host original**, y al cerrar el proceso que usaba ese namespace, este **queda destruido si no hay otros procesos que lo compartan**.

6. Usando el comando `unshare` crear un nuevo namespace de tipo Net.

1. `unshare -pid sh`
2. ¿Cuál es el PID del proceso `sh` en el namespace? ¿Y en el host anfitrión? Ayuda: los PIDs son iguales. Esto se debe a que en el nuevo namespace se sigue viendo el comando `ps` sigue viendo el `/proc` del host anfitrión. Para evitar esto (y lograr un comportamiento como los contenedores), ejecutar: `unshare --pid --fork --mount-proc`
3. En el nuevo namespace ejecutar `ps -ef`. ¿Qué sucede ahora?
4. Salir del namespace

Hacer `unshare --pid sh` **no crea un nuevo espacio aislado de `/proc`** , por lo que los PIDs no se verán aislados. El proceso `sh` se ejecuta en un nuevo PID namespace, pero sigue usando el `/proc` del host, así que los PID que ves son los del host.

¿Cuál es el PID del proceso `sh` en el namespace? ¿Y en el host anfitrión?

En la shell del namespace ejecutamos `echo $$` esto me dio el PID `13417`, en otra terminal ejecuto `ps -ef | grep sh` y entre todos los resultados encontramos `root 13417 12776 0 12:27 pts/5 00:00:00 sh`. En los 2 casos tienen el mismo PID, porque el proceso dentro del nuevo namespace todavía ve el `/proc` del sistema anfitrión.

Resultado del `ps -ef`

```
root@so:~# unshare --pid --fork --mount-proc bash
root@so:~# ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	12:36	pts/5	00:00:00	bash
root	2	1	23	12:36	pts/5	00:00:00	ps -ef

Ahora el proceso `bash` tiene PID 1, lo cual simula cómo funciona `init` en un contenedor o sistema aislado. Todo el espacio de procesos dentro del nuevo namespace empieza desde 1.

Terminamos saliendo del namespace con `exit` destruyéndolo si no hay otros procesos dentro.