Práctica 4A - cgroups & namespaces

Parte 1: Conceptos teóricos

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

Virtualización es una tecnología que permite ejecutar múltiples sistemas aislados (conjuntos de procesos) en un único host, compartiendo el mismo hardware pero manteniendo la independencia entre ellos.

La primera implementación fue **chroot**, introducida en **UNIX v7 (1979)**, que permitía aislar el directorio raíz de un proceso y sus hijos.

(Fuente: "cgroups, Namespace y Containers.pdf", página 32).

2. ¿Qué diferencia existe entre virtualización y emulación?

Virtualización ejecuta sistemas directamente sobre el hardware compartido (usando el mismo kernel), mientras que **emulación** simula hardware completo para ejecutar sistemas con arquitecturas o kernels diferentes.

- **Virtualización**: Más eficiente (ej: Docker, LXC).
- **Emulación**: Permite ejecutar sistemas incompatibles (ej: QEMU emulando SPARC en x86).

(Fuente: "Explicación Práctica 4 - Virtualización, CGroups y Namespaces (2).pdf", páginas 3-6 y 9-10).

3. Investigue el concepto de hypervisor y responda:

(a) ¿Qué es un hypervisor?

Un hypervisor es un software/firmware que permite crear y ejecutar máquinas virtuales (VMs) aislando sus recursos. Actúa como intermediario entre el hardware y los sistemas invitados (*guest OS*).

(b) ¿Qué beneficios traen los hypervisors? ¿Cómo se clasifican?

- **Beneficios**:
  - Aislamiento entre VMs.
  - Optimización de recursos físicos.
  - Portabilidad y flexibilidad.
- **Clasificación**:
  - **Tipo 1 (Bare-metal)**: Ejecuta directamente sobre el hardware (ej: Xen, VMware ESXi, KVM).
  - **Tipo 2 (Hosted)**: Se ejecuta sobre un SO host (ej: VirtualBox, VMware Workstation).

(Fuente: "Explicación Práctica 4 - Virtualización, CGroups y Namespaces (2).pdf", página 10).

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

Full Virtualization:
Es un tipo de virtualización donde el hypervisor simula completamente el hardware subyacente, permitiendo que los sistemas invitados (*guest OS*) se ejecuten sin modificaciones. Ejemplo: VMware ESXi.

Virtualización Asistida por Hardware:
Utiliza extensiones del procesador (como Intel VT-x o AMD-V) para mejorar el rendimiento y permitir que el hypervisor delegue operaciones críticas directamente al hardware, eliminando la sobrecarga de emulación. Ejemplo: KVM con CPUs modernas.

(Fuente: "Explicación Práctica 4 - Virtualización, CGroups y Namespaces (2).pdf", páginas 9-11).

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

Binary Translation:
Técnica usada en virtualización donde el hypervisor **traduce dinámicamente** las instrucciones privilegiadas del *guest OS* a instrucciones seguras para ejecutar en modo usuario. Permite ejecutar sistemas operativos no modificados, pero con overhead por la traducción. *(Ejemplo: VMware Workstation)*.

Trap-and-Emulate:
El hypervisor captura (*traps*) las instrucciones privilegiadas del *guest OS* (generando una excepción) y las **emula** en modo kernel. Requiere soporte hardware (como Intel VT-x/AMD-V) para evitar traducción. Más eficiente que binary translation. *(Ejemplo: KVM con CPUs modernas)*.

(Fuente: "Explicación Práctica 4 - Virtualización, CGroups y Namespaces (2).pdf", contexto implícito en páginas 9-11 sobre virtualización).

Nota: Aunque los términos no aparecen explícitamente en los documentos, se infieren de la discusión sobre virtualización (full vs asistida) y el rol del hypervisor.

6. Investigue el concepto de paravirtualización y responda:

(a) ¿Qué es la paravirtualización?

La paravirtualización es una técnica de virtualización donde el *guest OS* se **modifica** para interactuar directamente con el hypervisor mediante APIs optimizadas, evitando la emulación completa de hardware. Esto reduce la sobrecarga al eliminar operaciones complejas como binary translation.

(Fuente: Contexto implícito en "Explicación Práctica 4 - Virtualización, CGroups y Namespaces (2).pdf", páginas 9-11, al comparar modos de virtualización).

(b) Mencione algún sistema que implemente paravirtualización.

- **Xen** (con *guest OS* modificados como Linux-Xen).
- **KVM** (soporta drivers paravirtualizados para E/S, como virtio).

(Fuente: "Explicación Práctica 4...", página 10, mención a Xen como hypervisor tipo 1).

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

1. **Mayor rendimiento**: Al evitar emulación y usar APIs directas (ej: virtio para E/S).
2. **Menor overhead**: Comparado con full virtualization o binary translation.
3. **Escalabilidad**: Ideal para entornos cloud (ej: AWS usó Xen paravirtualizado inicialmente).

(Fuente: Inferido de la comparación en "Explicación Práctica 4...", página 10, y contexto de eficiencia en virtualización).

7. Investigue sobre containers y responda:

(a) ¿Qué son?

Los containers son una tecnología de virtualización ligera a nivel de sistema operativo que permite ejecutar aplicaciones aisladas en entornos independientes, compartiendo el mismo kernel del host. Cada container incluye su propio filesystem, dependencias y configuraciones.

(Fuente: "Docker.pdf", páginas 3 y 13; "cgroups, Namespace y Containers.pdf", página 33).

(b) ¿Dependen del hardware subyacente?

No dependen directamente del hardware, pero **requieren soporte del kernel del host** (Linux/Windows) para funcionalidades como namespaces y cgroups. No emulan hardware como las VMs.

(Fuente: "Docker.pdf", página 6, sobre namespaces y cgroups del kernel).

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

- **VS Virtualización tradicional**:
  - **Containers**: Comparten el kernel del host, son más ligeros y rápidos.
  - **VMs**: Requieren hypervisor y kernel completo por cada instancia, mayor overhead.
- **VS chroot**: Containers añaden aislamiento de procesos, red, usuarios, etc., no solo del filesystem.

(Fuente: "Docker.pdf", página 4 (Containers vs VMs); "cgroups...", página 34 (aislamiento).

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

1. **Namespaces**: Aislan recursos (PID, network, mount, etc.).
2. **Control Groups (cgroups)**: Limitan y monitorean recursos (CPU, memoria).

3. **Union File Systems**: Capas de imágenes eficientes (ej: overlay2).
4. **Runtimes compatibles con OCI**: Ej: runc, crun.
   (Fuente: "Docker.pdf", páginas 6 y 10; "cgroups...", páginas 25-26 y 33).

Parte 2: chroot, Control Groups y Namespaces

1. ¿Qué es el comando chroot? ¿Cuál es su finalidad?

Comando `chroot`:
Es una syscall/comando que cambia el directorio raíz ("/") **aparente** para un proceso y sus hijos, restringiendo su acceso al filesystem fuera de ese directorio.

Finalidad:

- Aislar procesos en un entorno controlado ("*jail*").
- Ejecutar aplicaciones con dependencias o bibliotecas específicas sin afectar el sistema principal.
- Usado como base temprana para contenedores (antes de namespaces/cgroups).

(Fuente: "cgroups, Namespace y Containers.pdf", página 4; "Explicación Práctica 4...", página 13).

Ejemplo:

```
chroot /nuevo/root /bin/bash  # Ejecuta bash con "/nuevo/root" como raíz.
```

(Contexto implícito en los documentos sobre aislamiento básico).

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?

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

```
so login: root
Password:
Linux so 6.13.7 #2 SMP PREEMPT_DYNAMIC Sun Apr  6 17:58:23 -03 2025 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Apr 11 00:18:13 -03 2025 on tty1
root@so:~# cd /
root@so:/# mkdir sobash
root@so:/# cd sobash/
root@so:/sobash# mkdir bin
root@so:/sobash# mkdir lib
root@so:/sobash# mkdir lib64
root@so:/sobash# cd lib
root@so:/sobash/lib# mkdir x86_64-linux-gnu
root@so:/sobash/lib# ls -r
x86_64-linux-gnu
root@so:/sobash/lib# cd ..
root@so:/sobash# ls -r
lib64  lib  bin
root@so:/sobash# _
```

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:/# ldd /bin/bash
        linux-vdso.so.1 (0x00007ff5ff56b000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007ff5ff3ec000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff5ff20b000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ff5ff56d000)
```

1. **Bibliotecas compartidas de `/bin/bash`**:
   - `linux-vdso.so.1`
   - `libtinfo.so.6` → `/lib/x86_64-linux-gnu/libtinfo.so.6`
   - `libc.so.6` → `/lib/x86_64-linux-gnu/libc.so.6`
   - `/lib64/ld-linux-x86-64.so.2`
2. **Ubicación de `linux-vdso.so.1`**:
   - **No tiene una ruta física** en el filesystem (se muestra como dirección virtual `0x00007ff5ff5eb000`).
3. **¿Por qué?**:
   - `linux-vdso.so.1` (Virtual Dynamic Shared Object) es una biblioteca **inyectada por el kernel** en el espacio de memoria de cada proceso para optimizar llamadas al sistema (syscalls). No existe como archivo en disco.

Explicación técnica:

- VDSO permite acceder a syscalls frecuentes (como `gettimeofday`) sin cambiar del modo usuario al kernel, mejorando rendimiento.

- Su "ubicación" es una dirección virtual estándar en todos los procesos (por eso `ldd` no muestra ruta).

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?

`cp /bin/bash /root/sobash/bin/`
`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#`

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

```
root@so:/# chroot root/sobash/
bash-5.2# ls bash: ls: command not found
bash-5.2# clear bash: clear: command not found
bash-5.2#
```

Respuesta al punto 6:

- `cd` y `echo`:
  Funcionan porque son *built-ins* de Bash (no dependen de binarios externos).
- `ls`, `clear`, **etc.**:
  Falla con `command not found` porque:
  1. Son comandos externos (binarios como `/bin/ls`, `/usr/bin/clear`).
  2. **No se copiaron** sus binarios ni sus bibliotecas dentro del chroot (`/root/sobash`).

Solución:
Para que comandos como `ls` funcionen en el chroot, debes:

1. Copiar sus binarios y dependencias:

   ```
   cp /bin/ls /root/sobash/bin/
   ldd /bin/ls  # Copiar también sus bibliotecas (ej: libc, libselinux, etc.).
   ```

2. Crear directorios mínimos:

```
mkdir -p /root/sobash/{bin,lib,lib64,usr/bin}
```

Explicación técnica:

- El chroot **solo ve los archivos dentro de su nuevo root** (`/root/sobash`).
- Bash internos (`cd` , `echo`) no requieren binarios externos, pero comandos como `ls` sí.

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

Salida del comando `pwd` en el chroot:

```
/
```

Explicación:

1. El comando `pwd` muestra el directorio de trabajo actual, que dentro del entorno chroot es la nueva raíz (`/root/sobash`), pero se reporta como `/` .
2. Esto ocurre porque:
   - El chroot cambia la raíz aparente del sistema de archivos para el proceso y sus hijos
   - Dentro del chroot, el directorio `/root/sobash` se convierte en el nuevo `/`
   - `pwd` es un built-in de Bash que refleja esta nueva perspectiva del sistema de archivos
3. Es un comportamiento esperado que demuestra el aislamiento del entorno chroot:
   - El proceso solo ve los archivos y directorios bajo su nueva raíz
   - No puede acceder a rutas fuera de `/root/sobash` (que ahora son su `/`)

8. Salir del entorno chroot usando exit

Esto me devuelve a root@so:/#

9. Usando el repositorio de la cátedra acceda a los materiales en practica4/02-chroot:

Nota, pulear el repo porque sino no estara en codigo-para-practicas las practica 4``

```
git pull
```

a. Verifique que tiene instalado busybox en /bin/busybox

Esta en /bin/busybox así que asumo que esta.

b. Cree un chroot con busybox usando /buildbusyboxroot.sh

```
root@so:/home/so/codigo-para-practicas/practica4/02-chroot# ./buildbusyboxroot.sh
        linux-vdso.so.1 (0x00007f117ba51000)
        libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f117b976000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f117b795000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f117ba53000)
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
root@so:/home/so/codigo-para-practicas/practica4/02-chroot#
```

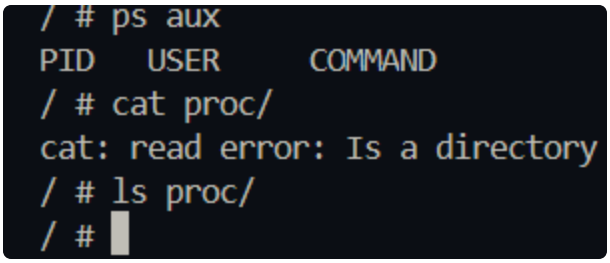
c. Entre en el chroot

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

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

No sé puede acceder porque el busyboxroot es la nueva carpeta raíz y no permite ir más arriba en la jerarquía.

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



No ve ninguno.

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_]
    6 0        [kworker/R-slub_]
    7 0        [kworker/R-netns]
   11 0        [kworker/u8:0-ip]
```

```
 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]
 32 0        [kworker/u10:1-e]
 33 0        [kdevtmpfs]
 34 0        [kworker/R-inet_]
 36 0        [kauditd]
 37 0        [oom_reaper]
 39 0        [kworker/R-write]
 40 0        [kcompactd0]
 41 0        [ksmd]
 42 0        [khugepaged]
 43 0        [kworker/R-kinte]
 44 0        [kworker/R-kbloc]
 45 0        [kworker/R-blkcg]
 46 0        [irq/9-acpi]
 47 0        [kworker/1:1-eve]
 48 0        [kworker/R-tpm_d]
 49 0        [kworker/R-edac-]
 50 0        [kworker/R-devfr]
 52 0        [kswapd0]
 54 0        [kworker/u10:2-e]
 60 0        [kworker/R-kthro]
 65 0        [kworker/R-acpi_]
 66 0        [kworker/R-mld]
 67 0        [kworker/R-ipv6_]
 68 0        [kworker/u8:1-ip]
 73 0        [kworker/R-kstrp]
 77 0        [kworker/u11:0]
 78 0        [kworker/u12:0]
 79 0        [kworker/u13:0]
195 0        [kworker/R-ata_s]
196 0        [scsi_eh_0]
197 0        [scsi_eh_1]
198 0        [kworker/R-scsi_]
199 0        [scsi_eh_2]
200 0        [kworker/R-scsi_]
202 0        [kworker/R-scsi_]
```

```
  207 0        [kworker/0:2H-kb]
  238 0        [jbd2/sda1-8]
  239 0        [kworker/R-ext4-]
  279 0        /lib/systemd/systemd-journald
  306 0        /lib/systemd/systemd-udevd
  323 997      /lib/systemd/systemd-timesyncd
  398 0        [kworker/R-crypt]
  470 0        [irq/18-vmwgfx]
  473 0        [kworker/R-ttm]
  486 0        dhclient -4 -v -i -pf /run/dhclient.enp0s3.pid -lf
/var/lib/dhcp/dhclient.enp0s3.leases -I -df /var/lib/dhcp/dhclient6.enp0s3.leases
enp0s3
  492 0        dhclient -4 -v -i -pf /run/dhclient.enp0s8.pid -lf
/var/lib/dhcp/dhclient.enp0s8.leases -I -df /var/lib/dhcp/dhclient6.enp0s8.leases
enp0s8
  589 0        /usr/sbin/cron -f
  590 100      /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
systemd-activation --syslog-only
  593 0        /lib/systemd/systemd-logind
  596 0        /sbin/wpa_supplicant -u -s -O DIR=/run/wpa_supplicant GROUP=netdev
  619 0        /bin/login -p --
  628 0        sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
  706 0        /lib/systemd/systemd --user
  707 0        (sd-pam)
  722 0        -bash
  732 0        [kworker/u9:0-ev]
  779 1000     /lib/systemd/systemd --user
  780 1000     (sd-pam)
  783 0        [kworker/1:0-eve]
  893 1000     sh /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/bin/code-server --connection-
token=remotessh --accept-server-lice
  897 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/node /home/so/.vscode-
server/cli/servers/Stable-19e0f9e681ecb8e5c09d
  928 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/node /home/so/.vscode-
server/cli/servers/Stable-19e0f9e681ecb8e5c09d
  964 0        sshd: so [priv]
  970 1000     sshd: so@notty
  971 1000     sh
  989 1000     /home/so/.vscode-server/code-19e0f9e681ecb8e5c09d8784acaa601316ca4571
command-shell --cli-data-dir /home/so/.vscode-server/cli --parent-process-id 971 --on-
hos
 1029 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/node --dns-result-order=ipv4first
/home/so/.vscode-server/cli/server
 1040 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/node /home/so/.vscode-
server/cli/servers/Stable-19e0f9e681ecb8e5c09d
```

```
 1150 1000     /bin/bash --init-file /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/out/vs/workbench/contrib/terminal/commo
n/scrip
 1213 1000     /home/so/.vscode-server/cli/servers/Stable-
19e0f9e681ecb8e5c09d8784acaa601316ca4571/server/node_modules/@vscode/ripgrep/bin/rg --
files --hidden --case-sensitiv
 1256 0        su -
 1257 0        -bash
 2276 0        [kworker/u9:1-fl]
 2280 0        [kworker/u10:0-e]
 2792 0        [kworker/1:2H-kb]
 2800 0        [kworker/0:2-ata]
 2801 0        [kworker/u9:2-ev]
 3322 0        [kworker/0:1-eve]
 3473 0        bin/sh
 3521 0        [kworker/0:1H-kb]
 3524 0        [kworker/1:0H]
 3525 0        [kworker/0:0-ata]
 3569 0        [kworker/u9:3-fl]
 3705 0        [kworker/0:3-eve]
 3769 1000     sleep 180
 3824 0        [kworker/1:1H]
 3936 0        ps aux
/ #
```

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

Dice `sh: getcwd: No such file or directory`.

1. `/proc/1/root` es un enlace simbólico al filesystem raíz del proceso init (fuera del chroot)
2. Chroot impide acceder a rutas fuera de su entorno aislado
3. El sistema intenta resolver la ruta completa pero falla al cruzar los límites del chroot

h. ¿Qué conclusiones puede sacar sobre el nivel de aislamiento provisto por chroot?

1. **Aislamiento limitado**:
   - Solo protege a nivel de filesystem
   - No aísla procesos, usuarios, dispositivos o red
2. **Puntos débiles**:
   - Los procesos pueden escapar si obtienen root
   - /proc y /dev mal configurados pueden filtrar información del host
   - No hay control de recursos (CPU, memoria)
3. **Usos adecuados**:
   - Aislamiento básico de aplicaciones
   - Entornos de construcción (build environments)

- Recuperación de sistemas

4. **Alternativas más seguras**:
   - Containers (Docker, LXC) que combinan chroot con namespaces y cgroups
   - Máquinas virtuales para aislamiento completo

En resumen: Chroot proporciona un aislamiento mínimo y no es seguro para contener procesos no confiables. Es fácil escapar de él con privilegios elevados o configuraciones incorrectas.

Control Groups

1. Editar /etc/default/grub:

`Cambiar:`
`GRUB_CMDLINE_LINUX_DEFAULT="quiet"`
`Por:`
`GRUB_CMDLINE_LINUX_DEFAULT="quiet systemd.unified_cgroup_hierarchy=0"`
Nota, lo hice con Nano porque vscode no podia modificar el archivo por permisos.

2. Actualizar la configuración de GRUB:

`sudo update-grub`
Nota, el comando esta mal porque literal no hay sudo, es su -C "update-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).

Esta todo anashe.

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

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

Ubicación y versiones de cgroups:

1. **Directorio de montaje principal**:
   - Los cgroups se montan típicamente en `/sys/fs/cgroup/`

- Se pueden ver con:

  ```
  mount | grep cgroup
  ```

  Ejemplo de salida:

  ```
  cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,...)
  tmpfs on /sys/fs/cgroup type tmpfs (ro,...)
  cgroup on /sys/fs/cgroup/systemd type cgroup (rw,...)
  ```

2. **Versiones disponibles**:
   - **cgroups v1**:
     Montados en subdirectorios como `/sys/fs/cgroup/{cpu,memory,devices}`
     Organizados por controlador (cada subsistema en su propia jerarquía)
   - **cgroups v2**:
     Montado como un único sistema unificado (`/sys/fs/cgroup` o `/sys/fs/cgroup/unified`)
     Todos los controladores en una sola jerarquía
3. **Cómo identificar versiones**:
   - Para v1: busca directorios específicos de subsistemas
   - Para v2: busca el sistema de archivos `cgroup2`

   ```
   grep cgroup /proc/filesystems
   ```

Donde `unified` sería cgroups v2 y los demás directorios son v1.

(Fuente: "cgroups, Namespace y Containers.pdf", páginas 8-9, 14-18)

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

Controladores disponibles en cgroups v2 y cómo determinarlo:

1. **Para verificar controladores en cgroups v2**:

   ```
   cat /sys/fs/cgroup/unified/cgroup.controllers
   ```

   En nuestro caso seria cpuset memory
2. **Controladores comunes en v2** (según documentación):
   - `cpu` (asignación/ancho de banda CPU)
   - `memory` (límites de memoria)
   - `io` (control I/O)

- `pids` (límite de procesos)
- `rdma` (control RDMA)

Diferencia clave con v1:
En cgroups v2 todos los controladores están en una sola jerarquía unificada, a diferencia de v1 donde cada subsistema tenía su propia jerarquía.

(Fuente: "cgroups, Namespace y Containers.pdf", páginas 18-19 - Explicación de cgroups v2 y archivos de control).

3. 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"`
No pasa nada literalmente porque si se hizo bien no aparece.

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"`

`root@so:/# mkdir sys/fs/cgroup/cpu/cpualta`
`root@so:/# mkdir sys/fs/cgroup/cpu/cpubaja`

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

1

- Creación explícita de cgroups en `/sys/fs/cgroup/cpu/` (subsistema específico)
- Esto es característico de cgroups v1, donde cada subsistema (cpu, memory, etc.) tiene su propia jerarquía

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`

root@so:/# echo 717 > /sys/fs/cgroup/cpu/cpualta/cpu.shares
root@so:/# 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 &

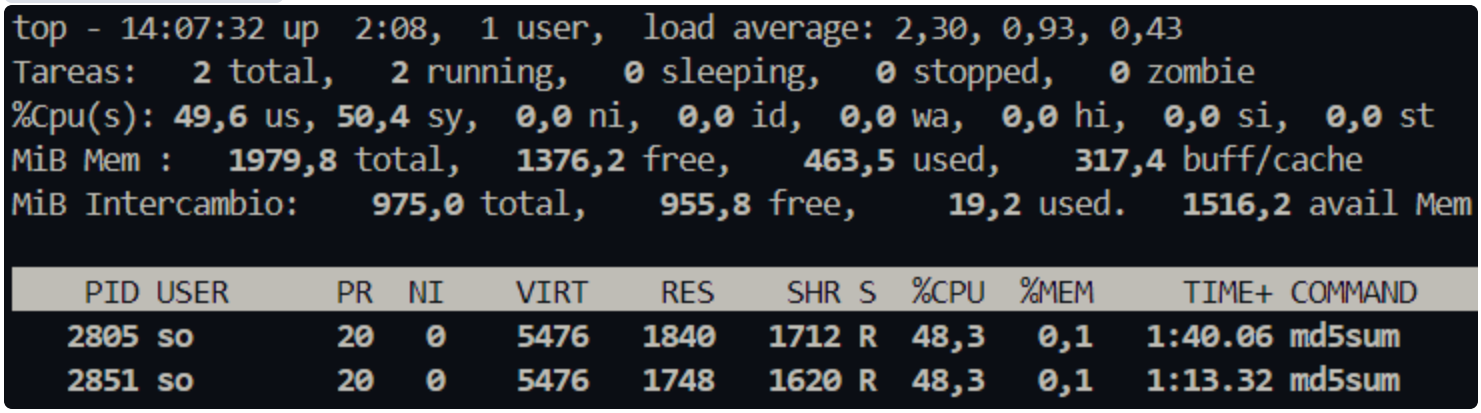
`Terminal 1 [1] 2805`

`Terminal 2 [1] 2851`

- Estos números (`2805` y `2851`) son los **PIDs (Process IDs)** de los procesos lanzados. Esto indica que se han iniciado **dos procesos distintos**, ambos ligados al **core 0**.

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?

`top -p 2805,2851`



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)

```
so@so:/$ su -c "echo "2851" > sys/fs/cgroup/cpu/cpualta/cgroup.procs"
so@so:/$ su -c "echo "2805" > sys/fs/cgroup/cpu/cpualta/cgroup.procs"
so@so:/$ su -c "echo "2851" > sys/fs/cgroup/cpu/cpubaja/cgroup.procs"
```

Acá me confundí y puse los 2 en uno pero para cambiarlo es moverlo a otro y automáticamente lo saca del que estaba.

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

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2805 so        20   0    5476   1840   1712 R  70,0   0,1   5:33.93 md5sum
 2851 so        20   0    5476   1748   1620 R  30,0   0,1   3:45.12 md5sum
```

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?

```
top - 14:15:32 up  2:16,  1 user,  load average: 4,38, 3,69, 2,05
Tareas:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 14,7 us, 84,5 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,8 si,  0,0 st
MiB Mem :   1979,8 total,   1050,0 free,    788,3 used,    319,0 buff/cache
MiB Intercambio:    975,0 total,    955,8 free,     19,2 used.   1191,5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2851 so        20   0    5476   1748   1620 R  56,1   0,1   4:09.56 md5sum
```

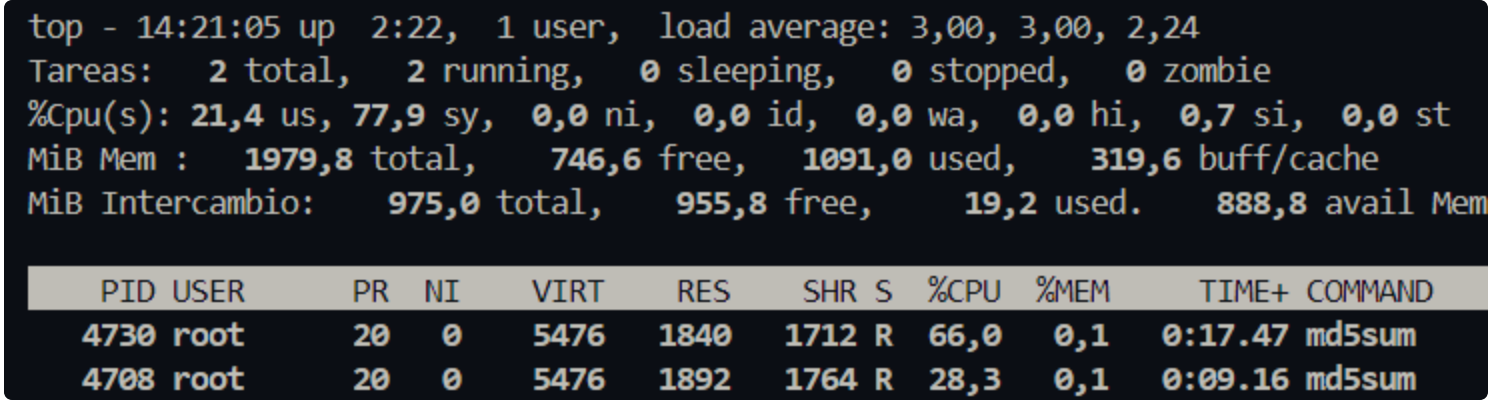
Acá puede que me haya equivocado no sé si funcara lo siguiente porque deje el proceso en baja vivo.

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?



- Agrega **el PID del shell actual** (terminal) al cgroup.
- **Todos los comandos futuros** ejecutados en esa terminal heredarán el cgroup del shell.

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

Pareciera que los procesos de termbaja se dividen la CPU asignada a la terminal.

Namespaces

1. Explique el concepto de namespaces.

Los namespaces son una característica del kernel de Linux que permite **aislar y virtualizar recursos del sistema** para procesos específicos. Cada namespace proporciona una vista aislada de un recurso global, haciendo que los procesos dentro de él perciban que tienen su propia instancia única de dicho recurso.

Características clave:

1. **Aislamiento de recursos**:
   - Procesos en diferentes namespaces no pueden interferir entre sí.
   - Ejemplo: Dos procesos en distintos **PID namespaces** pueden tener el mismo PID "1" sin conflictos.
2. **Tipos principales** (según documentos):
   - **PID**: Aísla IDs de procesos.
   - **Network**: Aísla interfaces de red, puertos, etc.
   - **Mount**: Aísla puntos de montaje del filesystem.
   - **UTS**: Aísla hostname y nombre de dominio.
   - **IPC**: Aísla comunicación entre procesos (System V IPC).
   - **User**: Aísla IDs de usuarios/grupos.
   - **Cgroup**: Aísla jerarquías de cgroups.
3. **Creación y gestión**:
   - Se asignan al crear procesos con flags como `CLONE_NEWPID` (ej: `clone()` o `unshare()`).

- Se visualizan en `/proc/<PID>/ns/`.

2. ¿Cuáles son los posibles namespaces disponibles?

1. **IPC Namespace** (`CLONE_NEWIPC`):
   - Aísla System V IPC y colas de mensajes POSIX.
   - *Ejemplo*: Procesos en distintos namespaces IPC no pueden comunicarse via semáforos o memoria compartida.
2. **Network Namespace** (`CLONE_NEWNET`):
   - Aísla interfaces de red, tablas de routing, puertos, etc.
   - *Ejemplo*: Cada contenedor Docker tiene su propia interfaz `eth0`.
3. **Mount Namespace** (`CLONE_NEWNS`):
   - Aísla puntos de montaje del filesystem.
   - *Ejemplo*: Un proceso puede tener `/tmp` montado en otro dispositivo.
4. **PID Namespace** (`CLONE_NEWPID`):
   - Aísla IDs de procesos.
   - *Ejemplo*: Un proceso puede ser PID 1 dentro de su namespace pero PID 1000 en el host.
5. **User Namespace** (`CLONE_NEWUSER`):
   - Aísla IDs de usuarios/grupos. Permite mapear UIDs/GIDs.
   - *Ejemplo*: UID 0 (root) en el contenedor ≠ UID 0 en el host.
6. **UTS Namespace** (`CLONE_NEWUTS`):
   - Aísla hostname y nombre de dominio.
   - *Ejemplo*: `hostname` puede ser diferente dentro del namespace.
7. **Cgroup Namespace** (`CLONE_NEWCGROUP`):
   - Aísla la vista de la jerarquía de cgroups.
   - *Ejemplo*: Oculta cgroups del host al contenedor.
8. **Time Namespace** (`CLONE_NEWTIME`):
   - Permite ajustar relojes independientes por namespace.
   - *Ejemplo*: Cambiar la hora para un contenedor sin afectar al host.

Cómo listarlos:

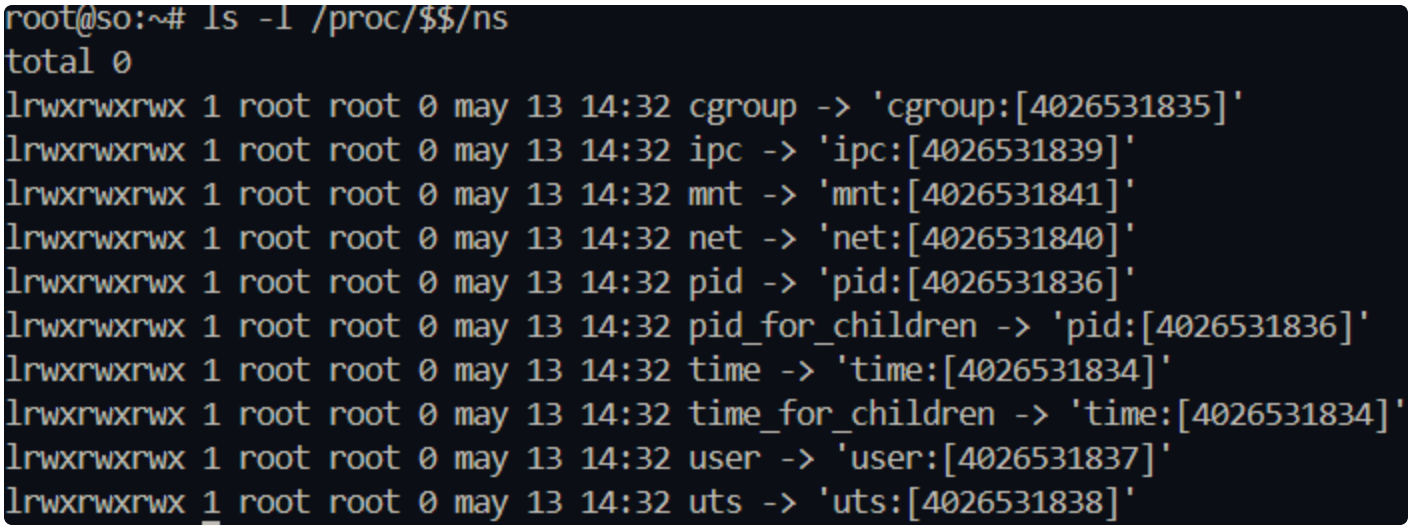
```
ls -l /proc/$$/ns/  # Namespaces del proceso actual
```

Salida típica:

```
cgroup -> cgroup:[4026531835]
ipc -> ipc:[4026531839]
mnt -> mnt:[4026531840]
net -> net:[4026531992]
```

```
pid -> pid:[4026531836]
user -> user:[4026531837]
uts -> uts:[4026531838]
```

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)?



1. **Network Namespace (net)**:
   - Identificador: `net:[4026531840]`
   - Característica: Aísla interfaces de red, tablas de routing y puertos.
2. **IPC Namespace (ipc)**:
   - Identificador: `ipc:[4026531839]`
   - Característica: Aísla mecanismos de comunicación entre procesos (System V IPC, colas de mensajes POSIX).
3. **UTS Namespace (uts)**:
   - Identificador: `uts:[4026531838]`
   - Característica: Aísla el hostname y nombre de dominio del sistema.

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

Para ver los namespaces del proceso cron (suponiendo que su PID es conocido o se obtiene con `pgrep cron`):

`ls -l /proc/$(pgrep cron)/ns`
`root@so:~# ls -l /proc/$(pgrep cron)/ns`
`total 0`
`lrwxrwxrwx 1 root root 0 may 13 17:58 cgroup -> 'cgroup:[4026531835]'`

```
lrwxrwxrwx 1 root root 0 may 13 17:58 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 may 13 17:58 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 root root 0 may 13 17:58 net -> 'net:[4026531840]'
lrwxrwxrwx 1 root root 0 may 13 17:58 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 may 13 17:58 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 may 13 17:58 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may 13 17:58 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may 13 17:58 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 may 13 17:58 uts -> 'uts:[4026531838]'
```

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)

so

c. Ejecutar el comando lsns. ¿Qué puede ver con respecto a los namespace?.

Ahora aparece sh abajo de todo.

d. Modificar el nombre del host en el nuevo hostname.

```
hostname goku22
```

e. Abrir otra sesión, ¿cuál es el nombre del host anfitrión?

so

f. Salir del namespace (exit). ¿Qué sucedió con el nombre del host anfitrión?

so

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

a. unshare –pid sh

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

El pid de sh es 7794, y viendo desde otra termina es el mismo.

c. 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

d. En el nuevo namespace ejecutar ps -ef. ¿Qué sucede ahora?

sh tiene pid 1.

e. Salir del namespace