Resumen Sistemas Operativos

Practica 1

Kernel de Linux

El kernel de GNU/Linux es un programa que ejecuta programas y gestiona dispositivos de hardware. Es el encargado de que el software y el hardware puedan trabajar juntos.

Las aplicaciones se comunican con el kernel mediante system calls.

Sus principales funciones son:

- Administración de memoria principal
- Administración de uso de la CPU

Arquitectura

El kernel de Linux es un kernel monolítico hibrido:

- Los drivers y el código del Kernel se ejecutan en modo privilegiado.
- Lo que lo hace híbrido es la posibilidad de cargar y descargar funcionalidad a través de módulos.

1- Tipo de kernel:

Kernel monolítico: todas las funciones del sistema operativo se ejecutan en el mismo espacio de memoria y comparten acceso directo a los recursos del hardware. No existe protección entre subsistemas del kernel; las funciones públicas pueden llamarse directamente entre diferentes subsistemas.
Ventajas: mucho más rápido pero limitado.
Desventajas: si hay un error en un driver BDS o kernel Panic.

Como se mencionó antes el kernel de Linux es monolitico.

Micro kernel: las grandes partes del kernel están protegidas entre sí y se ejecutan como servicios en el espacio de usuario. El kernel solo contiene el código necesario para comunicación (pasaje de mensajes). Esto implica implementar el planificador y la comunicación interprocesos (IPC), así como una gestión básica de memoria para establecer la protección entre aplicaciones y servicios.
Ventaja: el kernel es chiquito entonces es más fácil de entender, actualizar y optimizar.
Desventajas: baja performance.

No hay muchas implementaciones comerciales de este tipo de kernel.

2- Modulos:

Son fragmentos de código que se pueden cargar y descargar dinámicamente en el kernel en tiempo de ejecución. Se hablará de ellos más adelante.

3- Portabilidad:

El kernel de Linux es altamente portable y se ha adaptado a diversas arquitecturas de hardware, desde PCs hasta supercomputadoras.

Versiones

- 0.02 - 1991: La primer versión oficial.
- 0.12 - 1992:. Se cambia la licencia a una GNU.
- 1.0. - 1994: Todos los componentes están maduros.
- 2.0. - 1996: Se define un sistema de nomenclatura.
    - 2.4. - 2001: Linux ya era robusto y estable. Se deja de desarrollar en 2010.
    - 2.6 - 2006. Muchas mejoras del kernel: los threads, mejoras de planificación y soporte del nuevo hardware.

- 3.0. No hay muchos cambios, se decidió cambiar por los 20 años del SO y no superar los 40 números de revisión, fue un cambio estético. Es compatible con 2.6; sin embargo introdujo mejoras de gestión de energía, mejoras de virtualización, mejoras de rendimiento de red y de soporte de hardware.
- 4.0. Permite aplicar parches y actualizaciones sin reiniciar el SO. Soporta nuevas CPU. Este cambio incorporó mejoras en administración de memoria, seguridad, gestion de archivos y soporte de hw.
- 5.0. incorpora varias cosas en distintas versiones:
    - scheduling que disminuye el consumo energético en smartphones.
    - soporte de namespaces para Android.
    - soporte de archivos swap en BTRFS.
    - kernel lockdown modo que previene el acceso de procesos de usuario a la memoria del kernel (5.4).
    - soporte a USB 4 (5.6).
    - nuevo mecanismo para manejar syscalls de Windows (5.11).
    - Landlock security module que permite restringir acciones que un conjunto de programas pueden ejecutar en un filesystem (5.13).
    - system call para crear áreas de memoria secretas (5.14).
    - soporte inicial para IPv6 (5.19).
- 6.0. incorpora varias cosas en distintas versiones:
    - soporte módulos escritos en Rust (6.1).
    - mejoras en la confiabilidad de RAID5/6 en Btrfs (6.2).
    - mejoras en el rendimiento y la fragmentación de Btrfs (6.3).
    - soporte inicial para USB4.0 v2 (6.5).
    - nuevo scheduler de tareas (EEVDF) que mejora la equidad en comparación con el algoritmo anterior (CFS) (6.6).
    - soporte para un nuevo sistema de archivos, BCachefs, con características similares a ZFS y Btrfs (6.7).
    - optimización de estructuras de datos para networking, mejorando la performance de TCP hasta un 40% con muchas conexiones concurrentes (6.8).

Versionado

- Versión < 2.6: X.Y.Z
  - X: serie principal. Funcionalidades importantes.
  - Y: indicaba si era de producción o desarrollo.
    - números Y pares indicaban producción (estable) y los Y impares indicaban una versión en desarrollo.
  - Z: bugfixes.
- Versión ≥ 2.6 y < 3.0: A.B.C.[D]
  - A: Denota versión. Cambia con menor Frecuencia (cada varios años).
  - B: Denotan revisión mayor, cambios importantes.
  - C: Denota revisión menor. Cambia cuando hay nuevos drivers o características.
  - D: Se corrige un error grave sin agregar funcionalidad.
- Versión ≥ 3.0: A.B.C[-rcX]
  - A Denota revisión mayor. Cambia con menor Frecuencia (cada varios años).
  - B Denota revisión menor. Solo cambia cuando hay nuevos drivers o características.
  - C Número de revisión.
  - rcX Versiones de prueba

Compilado

Se compila para:

- Soportar nuevos dispositivos como, por ejemplo, una placa de video
- Agregar mayor funcionalidad (soporte de nuevos filesystems)
- Optimizar funcionamiento de acuerdo al sistema en el que corre
- Adaptarlo al sistema donde corre (quitar soporte de hardware no utilizado)
- Corrección de bugs (problemas de seguridad o errores de programación)

Comandos útiles

make menuconfig:

- Inicia la interfaz de configuración menuconfig

make clean:

- Limpia el árbol de código fuente del kernel eliminando archivos generados durante compilaciones anteriores.

make (con parámetro -j):

- Compila el kernel utilizando directivas del archivo Makefile.
- Puede durar mucho tiempo dependiendo del procesador que se tenga.
- El parámetro -j especifica el número de trabajos paralelos durante la compilación.

make modules:

- Compila únicamente los módulos del kernel. Actualmente se incluye en la compilación principal este paso.

make modules_install:

- Instala los módulos compilados en /lib/modules/<kernel-version>/,

make install:

- Instala el kernel compilado en el sistema.
- Copia el kernel y archivos necesarios a /boot y actualiza la configuración del cargador de GRUB

Necesidades para compilación

- gcc: Compilador de C

- make: ejecuta las directivas definidas en los Makefiles
- binutils: assembler, linker
- libc6: Archivos de encabezados y bibliotecas de desarrollo
- ncurses: bibliotecas de menú de ventanas (solo si usamos menuconfig)
- initrd-tools: Herramientas para crear discos RAM

En Debian y distribuciones derivadas, todo este software se encuentra empaquetado

```
# apt-get install build-essential libncurses-dev
kbuild flex bison libelf-dev bc
```

Proceso compilación

1. Obtener el código fuente.

   ```
   $ cd /usr/src
   $ sudo wget https://kernel.org/pub/linux/kernel/v6.x/linux-6.7.tar.xz
   ```

   Por convención se guarda en /usr/src.

   Opcionalmente se puede descargar un archivo con una firma criptográfica del kernel descargado que comprueba que es el mismo que esta subido.

2. Preparar el árbol de archivos del código fuente.

   ```
   $ mkdir $HOME/kernel
   $ cd $HOME/kernel
   $ tar xvf /usr/src/linux-6.7.tar.xz
   ```

   /usr/src no tiene permisos de escritura para users no privilegiados, entonces descomprimo en uno donde si se tengan permisos. Generalmente se crea un enlace simbólico llamado linux apuntando al directorio del código fuente que actualmente se está configurando `$ ln -s /usr/src/linux-6.7 /usr/src/linux`

   Se puede “emparchar” el código para actualizarlo a través del comando patch:

   - Mecanismo que permite aplicar actualizaciones sobre una versión base. Se basa en archivos diff, que indican que agregar y qué quitar.
   - Hay de 2 tipos:
     - No incrementales: Se aplican sobre la versión mainline anterior (X.0.0). Cómo el que usarán en la práctica.
     - Incrementales: Se aplican sobre la versión inmediatamente anterior.

- Mas sencillo descargar el archivo diff y aplicarlo en vez de descargar todo el código de la nueva versión.

```
$ cd linux; zcat ../patch-6.8.gz | patch -p1
```

```
$ xzcat $HOME/kernel/patch-6.8.xz | patch -p1
```

3. Configurar el kernel
   - Mediante el archivo .config que está en el directorio donde está el código fuente del Kernel.
   - Contiene instrucciones de que es lo que el kernel debe compilar.
   - Se puede ver la configuración de los kernels instalados en /boot:
   - Hay tres interfaces que permiten generar este archivo
     - make config:
       - Interfaz: Basada en texto, sin menús ni ayuda contextual.
       - Requisitos: Terminal y sistema de compilación básico.
     - make menuconfig:
       - Interfaz: Basada en texto con menús, utilizando ncurses para una interfaz con paneles en la terminal.
       - Requisitos: Terminal y sistema de compilación básico, incluye herramientas GNU (gcc y make).
     - make xconfig:
       - Interfaz: Gráfica, requiere soporte para X Window System.
       - Requisitos: Sistema con entorno gráfico y bibliotecas necesarias para desarrollo gráfico.
   - Lo ideal es ir manteniendo el .config para no tener que configurar todo de cero (tedioso)
     - Cada nueva versión, puede valerse de un .config anterior.
       - Copiar y renombrar configuración:
       - Con olddefconfig se toma la configuración antigua y la actualiza con valores por defecto para las nuevas opciones.
4. Construir el kernel a partir del código fuente e instalar los módulos.
   - Compilar kernel con
     - $ make -jX # X es el número de threads.

        - Incluye compilacion de todos los módulos necesarios para satisfacer las opciones que hayan sido seleccionadas como módulo.
    - Verificar que este proceso no arroje errores al concluir.
5. Reubicar el kernel e instalarlo (junto con los módulos)
    - Recién en esta parte hay que convertirse en root.
    - Al terminar el proceso de compilación, la imagen del kernel quedará ubicada en directorio-del-código/arch/arquitectura/boot/. Debe ser reubicada en el directorio /boot para la instalación. También deben ser reubicados e instalados los módulos.
        - Con el siguiente comando se realiza esto automáticamente y se instalan los módulos y el kernel
            - make modules_install
            - make install
6. Creación del initramfs

    initramfs
    - Sistema de archivos temporal que se monta durante el arranque del sistema.
    - Proporcionar un entorno mínimo y temporal que permite al SO cargar los controladores necesarios, montar el sistema de archivos raíz real y completar el proceso de inicio.
    - Contiene ejecutables, drivers y módulos necesarios para lograr iniciar el sistema. Luego del proceso de arranque el disco se desmonta

    Se realiza automáticamente con los comandos mencionados en el punto anterior.
7. Configurar y ejecutar el gestor de arranque (GRUB en general).

    El gestor de arranque no tiene conocimiento automático cuando se tiene un nuevo kernel y, por lo tanto, no lo incluirá en la lista de opciones de arranque del sistema. Por lo tanto, es necesario reconfigurar el gestor de arranque para que se incluya el nuevo kernel.

    Se realiza automáticamente con los comandos mencionados en el punto anterior.
8. Reiniciar el sistema y probar el nuevo kernel.

    $ reboot

Característica Energy-aware Scheduling

- En procesadores ARM big.LITTLE:

    - Dos tipos de núcleo:
        - Núcleos potentes de alto rendimiento (big). Se usan para tareas intensivas en recursos.
        - Núcleos menos potentes pero con mucho menor consumo (LITTLE). Se usan para tareas menos exigentes.
        - Permite al sistema balancear carga de trabajo entre núcleos en base a demandas de rendimiento y energía actuales.
- Con esta característica habilitada cuando se despierta un proceso, el scheduler lo asigna al núcleo que mejor se adapte a las necesidades de rendimiento y consumo de energía en ese momento.
    - Si el proceso requiere un alto rendimiento, podría asignarse a un núcleo big.
    - Si requiere menos recursos, podría asignarse a un núcleo LITTLE para conservar energía.
- Beneficia a dispositivos móviles para equilibrar el rendimiento y la duración de la batería.

System call memfd_secret()

Permite crear áreas de memoria secretas que no pueden ser accedidas ni por el usuario root. Esa memoria se puede utilizar para almacenar claves criptográficas u otros datos que no deben ser expuestos a otros. El kernel no puede acceder al contenido de regiones de memoria creadas con esta system call.

Practica 2

System Call

Una System Call es un llamada al kernel para ejecutar una función específica que controla dispositivo o ejecuta instrucción privilegiada. Su propósito es proveer una interfaz común para lograr portabilidad y su funcionalidad se ejecuta en modo Kernel pero en contexto del proceso.

POSIX APIs

Los SO proveen interfaces para que los procesos accedan a un conjunto de funciones. En UNIX lo hace libc. Gran parte de la funcionalidad de la libc y de las system calls está definida por el estándar POSIX.

Libc provee librerias de C y es una interfaz entre aplicaciones de usuario y las syscalls.

En el caso de las System Calls, el desarrollador generalmente interactúa con la API y no directamente con el Kernel. En UNIX por lo general cada función de la API se corresponde con una System Call. Por lo tanto, el flujo de trabajo es: aplicación → librería de C → Kernel (System Call).

Invocaciones

Se puede hacer usando un wrapper de glibc o invocando la macro syscall. syscall es una función proporcionada por glibc para hacer system calls de forma explícita. Mas difícil de usar y menos portable que wrapper pero más fácil y portable que assembler. syscall es más útil cuando se trabaja con system calls que son especiales o más nuevas que la glibc que se está usando.

```c
#include <stdlib.h>
#include <sys/syscall.h>
#include <sys/time.h>
#include <unistd.h>
#define SYS_gettimeofday 78
void main(void){
    struct timeval tv;
    /* usando el wrapper de glibc */
    gettimeofday(&tv, NULL);
    /* Invocación explícita del wrapper general */
    syscall(SYS_gettimeofday, &tv, NULL);
}
```

Parámetros:

- Numero system call y parámetros para la system call.

Interrupciones

Manera ejecutar syscalls depende procesador. El x86 las maneja con interrupciones. Cuando hay una interrupción el kernel trata de manejarla para garantizar que el programa pueda continuar sin perder datos.

libc provee una abstracción:

- Salva el índice de la system call y sus argumentos en los registros correspondientes
- Invoca la instrucción necesaria para invocar la syscall.
  - A través de la estructura syscall table y el índice se determina que handler function invocar.
- Recupera el valor retornado por la syscall.

Desarrollo System Call

1. Se identifican con un número único: syscall number.
2. Se agrega una syscall a la syscall table. En el archivo /arch/x86/syscalls/syscall_32.tbl para la arquitectura de x86 de 32 bits o /arch/x86/syscalls/syscall_64.tbl para la arquitectura de x86 de 64 bits.
   - Debemos considerar el sys call number.
   - Respetar las convenciones del Kernel.
3. La syscall se debe declarar, los parámetros son realizados por el stack. Para informarle al compilador de los parámetros se hace mediante la macro asmlinkage quien instruye al compilador a pasar parámetros por stack y no por registros.
4. Debemos definir la syscall en algún punto del árbol de fuentes. Se puede usar algún archivo existente o se puede incluir un nuevo archivo y su correspondiente Makefile.
5. Finalmente se recompila el kernel.

Strace

Strace es un comando que reporta las syscalls que realiza cualquier programa. Utilizándolo junto al parámetro -f tiene en cuenta, además, a los procesos hijos.

Módulos

La porción de código que se agrega al kernel en tiempo de ejecución se denomina "modulo".

- Permiten extender o modificar el kernel sin necesidad de reiniciar el sistema completo.
  - Si no se hiciera esto el kernel debería ser 100% monolitico
    - Los módulos en si serian parte del kernel, por lo que, si se desea agregar una funcionalidad, se debería re-compilar el kernel, instalarlo y rebotear la PC.
- Los controladores de dispositivos, sistemas de archivos y otras funcionalidades se implementan como módulos en Linux.
- Los módulos disponibles se ubican en /lib/modules/version del kernel.

Los comandos principales para el manejo de módulos del kernel son:

1. modprobe: carga un módulo del kernel en la memoria en tiempo de ejecución. Usa la información generada por depmod e información de /etc/modules.conf para cargar el módulo especificado. El archivo /lib/modules//modules.dep es utilizado por el comando modprobe en GNU/Linux para conocer las dependencias de los módulos del kernel que se deben cargar cuando se carga un módulo determinado.
2. insmod: carga un módulo del kernel en la memoria. No maneja automáticamente las dependencias del módulo.
3. rmmod: se utiliza para eliminar (descargar) un módulo del kernel de la memoria.
4. lsmod: lista los módulos del kernel que están actualmente cargados en la memoria del sistema. Equivalente a: cat /proc/modules.
5. depmod: Este comando se utiliza para generar y manipular los archivos de dependencia de los módulos del kernel. Permite calcular las dependencias de un

módulo. Junto al parámetro -a escribe las dependencias en el archivo /lib/modules/version/modules.emp .

6. modinfo : muestra información sobre el módulo.

Crear modulo

Para crear un módulo se deben proveer dos funciones: inicialización (insmod) y descarga (rmmod).

```
#include <linux/module.h>
#include <linux/kernel.h>
int init_module(void){
        printk(KERN_INFO "Hello world 1.\n");
return 0;
}

void cleanup_module(void){
        printk(KERN_INFO "Goodbye world 1.\n");
}
```

También puede indicarle otras funciones las cuales sirven para personalizar los nombres de las funciones de inicialización y cleanup: module_init() y module_exit(). Estas realizan la vinculación entre los nombres personalizados e init_module() y cleanup_module(). Para ver la información del log que arrojan se puede usar dmesg.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
static int hello_init(void){
        printk(KERN_INFO "Hello! \n");
return 0;}

static void hello_exit(void){
printk(KERN_INFO "Goodbye! \n");}

module_init(hello_init);
module_exit(hello_exit);
```

Los parámetros se definen con la macro module-param . Siendo name el nombre del parámetro expuesto al usuario y de la variable que contiene el parámetro en nuestro módulo, type el tipo (byte, short, ushort, int, uint, long, etc.) y perm los permisos al archivo correspondiente al módulo el sysfs.

```
static char *user_name = "";
module_param(user_name, charp, 0);
MODULE_PARM_DESC(user_name,"user name");
```

Al cargar el módulo indicamos el valor del parámetro:

$ sudo insmod hello.ko user_name=agus

Luego se construye el Makefile (archivo que se incluye en la carpeta raíz de un proyecto para que le digan a un programa, make, que se ejecuta desde la terminal, qué hacer con cada uno de los archivos de código para compilarlos)

```
obj-m += hello.o
```

Y luego se compila

```
$ make -C <KERNEL_CODE> M=$(pwd) modules
```

(clean: make -C /lib/modules/$(shell uname -r)/build M=$(pwd) clean)

Cuando se compila se generan distintos archivos:

- nombre.c: es el archivo fuente en C del módulo del kernel. Contiene el código fuente del módulo que se compiló para producir los archivos objeto y el archivo del módulo.
- nombre.ko: es el archivo binario final, que incluye estructuras de datos generadas automáticamente, necesarias para que el Kernel pueda cargar el módulo
- nombre.mod: almacena la ruta absoluta del archivo nombre.o compilado

- nombre.mod.c: contiene llamadas a macros que le proveen la información del módulo al Kernel
- nombre.mod.o: es el archivo binario compilado a partir de nombre.mod.c
- nombre.o: Es el archivo binario compilado a partir de nombre.mod.c
- Module.symvers: Contiene una lista de todos los símbolos exportados al compilar un Kernel
- modules.order: cuando se compilan múltiples módulos a la vez, lista en qué orden se fueron generando los nombre.ko

Con $ insmod nombre.ko agrego el módulo en tiempo de ejecución (se invocaría la funcion de carga) y con $ rmmod nombre.ko lo descargo den tiempo de ejecución.

MODULE_LICENSE

Permite que los módulos cargables del kernel declaren su licencia. No es obligatoria, pero el kernel será marcado como “contaminado”.

Dispositivos y Drivers

Dispositivos

Es cualquier dispositivo de hardware. Cada operación sobre un dispositivo se hace con un código especial para ese dispositivo llamado “driver” y se implementa como un módulo. Los dispositivos se pueden clasificar en dos:

- Dispositivos de acceso aleatorio:
  - Almacenan y recuperan datos en bloques de tamaño fijo.
  - Acceso no secuencial
  - Ejemplos: discos duros (HDD), unidades de estado sólido (SSD), unidades USB y tarjetas de memoria.
- Dispositivos seriales:
  - Transmiten datos secuenciales, uno tras otro, en forma de caracteres.
  - Ejemplos: mouse, teclado, dispositivos de sonido (altavoces, micrófonos), GPS, etc.

Un dispositivo se identifica con el major (identifica tipo de dispositivo) y minor number (identifica instancia). Estos se encuentran en kernel_code/linux/Documentation/devices.txt

Cada dispositivo de hardware es un archivo (device file), y accedemos a ellos mediante operaciones básicas (espacio kernel): read , write y ioctl. Si leemos/escribimos desde él (/dev) lo hacemos sobre datos "crudos" del disco (bulk data).

Por convención están en el /dev, donde hay archivos de:

- Dispositivos de caracteres.
- Dispositivos de bloques.
- Pseudo-Dispositivos: no se corresponden con dispositivos de hardware, si no que proporcionan varias funciones útiles.
- Enlaces simbólicos.

Los archivos de dispositivos se crean mediante el comando mknod

```
mknod[- m<mode >] file[b|c ]major  minor
```

Se elige b o c según es un dispositivo de carácter o de bloque. El minor y el major number lo obtenemos del txt de devices.

Es necesario decirle al kernel qué hacer cuando se escribe el device file y qué hacer cuando se lee desde ese device file. Todo esto se realiza en un módulo (drivers).

Driver

Software que actúa como intermediario entre el SO y el hardware. Su función principal es permitir que el sistema operativo se comunique con los componentes físicos ofreciéndole abstracción respecto a los mismos (ofrece portabilidad).

Si se tiene un módulo para tener un driver completo faltaría la interacción con un dispositivo físico o virtual.

Existen distintos tipos de drivers:

- Drivers de bloques: son un grupo de bloques de datos persistentes. Leemos y escribimos de a bloques, generalmente de 1024 bytes.
- Drivers de carácter: Se accede de a 1 byte a la vez y 1 byte sólo puede ser leído por única vez.
- Drivers de red: tarjetas ethernet, WIFI, etc.

Creación Drivers

Mediante la struct file operations se especifica que funciones leen/escriben al dispositivo. Cada variable posee un puntero a las funciones que implementan las operaciones sobre el dispositivo.

```
struct file_operations my_driver_fops = {
read: myDriver_read,
write: myDriver_write,
open: myDriver_open,
release: mydriver_release};
```

En la función module init registro mi driver.

```
register_chrdev(major_number, "myDriver", &my_driver_fops);
```

Se le debe pasar el major number (dispositivo), nombre del driver y la file_operation

En la función module exit desregistro mi driver

```
unregister_chrdev(major_number, "myDriver");
```

Se le debe pasar el major number y el nombre del driver.

Escritura del archivo de dispositivo

```
ssize_t myDriver_write(struct file *filp, char *buf,size_t
   count, loff_t *f_pos);
```

Lectura del archivo de dispositivo

```
ssize_t myDriver_read(struct file *filp, char *buf,size_t
   count, loff_t *f_pos)
```

Parámetros de las funciones funciones:

- struct file: Estructura del kernel que representa un archivo abierto.
- char *buf: El dato a leer o a escribir desde/hacia el dispositivo(espacio de usuario)
- size t count: La longitud de a leer de buf.
- loff t *f pos: La posición actual en el archivo

Alternativa /proc

- Sistema de ficheros virtual
- No ocupa espacio en disco
- Al leer o escribir en un archivo de este sistema del /proc se ejecuta una función del kernel que devuelve o recibe los datos
    - Lectura: read callback
    - Escritura: write callback
- En Linux, /proc muestra información de los procesos, uso de memoria, módulos, hardware, etc.
- Mecanismo de interacción entre el usuario y el kernel
    - Los módulos pueden crear entradas /proc para interactuar con el usuario

Pasos para el uso de este mecanismo:

1. Crear un módulo del kernel con funciones init module() y cleanup module()
2. Definir variable global de tipo struct file operations
    - Especifica qué operaciones en el /proc se implementan y su asociación con las funciones del módulo

```
struct file_operations fops = {
.read = myproc_read,
.write = myproc_write,
};
```

3. En la función de inicialización, crear la entrada del /proc con la función proc create():

```
struct proc_dir_entry *proc_create(const char *name,
   umode_t mode,struct proc_dir_entry *parent,const
   struct file_operations *ops);
```

   - Parámetros:
     - name: nombre de la entrada
     - mode: mascara octal de permisos (p.ej., 0666)
     - parent: puntero al directorio padre (NULL → directorio
     - ops: Puntero a la estructura que define las operaciones

4. En la función cleanup del módulo, eliminar la entrada /proc creada

```
void remove_proc_entry(const char *name,struct
   proc_dir_entry *parent);
```

Módulos y drivers en initramfs

Este se genera o actualiza:

- Durante la instalación del sistema operativo.
- Después de instalar un nuevo kernel.
- Cuando se realizan cambios significativos en la configuración del sistema.

Los módulos y drivers que deberá tener un initramfs mínimamente para cumplir su objetivo serán los necesarios para:

- Montar el sistema de archivos raíz (root filesystem).
- Detectar y montar dispositivos de almacenamiento.
- Detectar y cargar los controladores necesarios para el hardware crítico del sistema.

Practica 3

- Las aplicaciones necesitan almacenar y recuperar información. La información se almacena en archivos.
- File System es la parte del SO que se encarga del manejo de los archivos.
- Un archivo es un conjunto de datos que va a vivir mucho tiempo probablemente. Para el usuario es un conjunto de datos persistente pero para el SO son bloques de disco.
- Sus componentes son: nombre, metadata y datos.

File system

- Estructura jerárquica (árbol de archivos) de archivos y directorios que permite la creación, eliminación, modificación y búsqueda de archivos y su organización en directorios.
- Administra el control de acceso a archivos y espacio en discos asignados a él.
- Operan sobre bloques de datos (conjuntos consecutivos de sectores físicos).
- Define convenciones para el nombrado de archivos.
- No siempre es necesario tener un file system para acceder a un disco o partición, si no se lo tiene algunos SO lo tratan como un bloque de datos sin estructura específica.

Area de swap Linux

Partición especial del disco duro que se utiliza como memoria virtual. Cuando la RAM se agota, el sistema operativo Linux puede transferir datos de la RAM a esta área de swap para liberar espacio de esta.

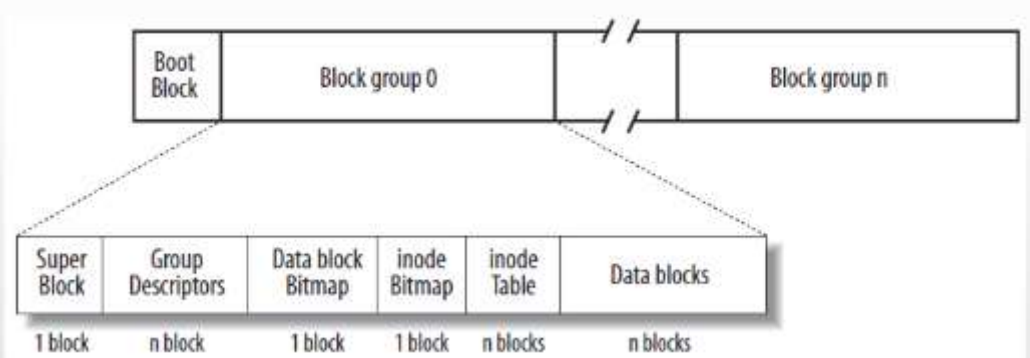
lost+found de Linux

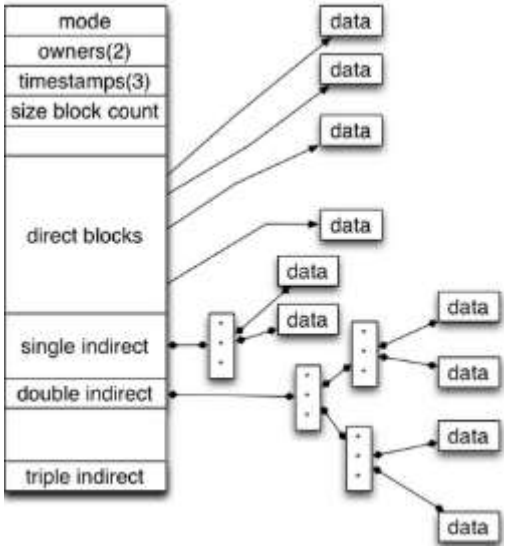
- Almacena archivos encontrados en el SO durante el proceso de verificación de integridad o reparación del sistema de archivos.

- Cuando el file system se corrompe, el SO puede intentar repararlo durante el próximo inicio. Durante este proceso, el file system puede encontrar archivos que no están asociados con ningún directorio o con algún otro problema.
- Estos archivos se pueden recuperar (tras una revisión del file system a través de la herramienta fsck) y mover al directorio "lost+found" para que el admin del sistema los revise manualmente y decida qué hacer con ellos.

Ext2

- Primer sector de la partición no es administrado por Ext2
- Es un filesystem extendido. Dividido en “block groups”, todos son del mismo tamaño menos el último.
- Los “block groups” reducen la fragmentación y aumentan la velocidad de acceso.
- “Superblock” y “Group Descriptors Table” replicados en block groups para backup
    - No es necesario replicarlos en todos los grupos.
- Por cada bloque existe un “Group Descriptor”.
- “Block Bitmap” e “Inode Bitmap” indican si un bloque de datos o inodo está libre u ocupado.
- Un inodo es una estructura de datos de los sistemas de archivos. Cada archivo del file system es representado por un inodo. Los inodos contienen metadata de los archivos y punteros a los bloques de datos.
    - Metadata: permisos, owner, grupo, flags, tamaño, numero bloques usados, tiempo acceso, cambio y modificación, etc.
- Tabla de inodos consiste de una serie de bloques consecutivos donde cada uno tiene un numero predefinido de inodos. Todos los inodos de igual tamaño: 128 bytes (por default).
- Solo “superblock” del grupo 0 se lee y se modifica
- El nombre del archivo no se almacena en el inodo. Los atributos extendidos, como las ACLs, se almacenan en un bloque de datos.
- Datos se almacenan en bloques de 1024, 2048 o 4096 bytes. Esto es elegible al momento de generar el file system, no se puede modificar.





Ext3

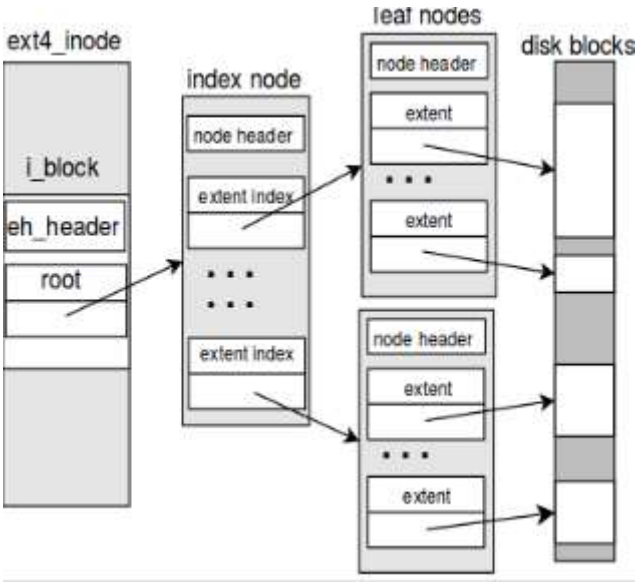
- Evolución de Ext2 y es compatible con él.
- El tamaño máximo de archivo es 2TB. El tamaño máximo de Filesystem es 32 TB.
- La cantidad máxima de subdirectorios es 32000. Permite extensión online del file system.
- Su principal mejora se basa en la incorporación del “journaling”, que permite reparar posibles inconsistencias en el file system.
    - El journaling mantiene un journal o log de los cambios que se están realizando en el file system. Permite una rápida reconstrucción de file systems corruptos.
    - Tres niveles configurables:

        - Writeback: mayor riesgo. Solo se graban los metadatos. Datos podrían ser grabados antes o después que el journal sea actualizado.
        - Ordered: riesgo mediano. Solo graba los metadatos. Garantiza grabar el contenido de los archivos a disco antes que hacer commit de los metadatos en el journal.
        - Journal: metadatos y datos son escritos en el journal antes de ser grabados en el file system principal.
    - Implica muchas escrituras en el disco.

Ext4

- Es la evolución de Ext3. Mejora compatible de ext3.
- Es un sistema de archivos de 64 bits. La cantidad máxima de subdirectorios es 64000, e incluso es extensible.
- El tamaño del inodo es 256 bytes.
- Usa extents
    - Descriptor que representa un rango contiguo de bloques físicos.
    - Por ejemplo, un archivo de 100 MB puede asignarse a una única extensión en lugar de 25600 bloques individuales.
    - Los archivos grandes se dividen en múltiples extents.
    - Las extents mejoran el rendimiento y reducen la fragmentación al fomentar diseños continuos en el disco.
    - Cada extent puede representar 215 bloques (128MB con bloques de 4KB, 4 extents por inodo). Para archivos más grandes, se utiliza un árbol de extents.
- Tiene mejor alocación de bloques para disminuir la fragmentación e incrementar el rendimiento: “persistent preallocation”, “delay and multiple block allocation”.
    - Multiblock allocation:
        - Se asignan muchos bloques en una sola llamada, en lugar de un solo bloque por llamada (mballoc asignador multibloque)
    - Delay allocation:
        - Posterga la asignación de bloques tanto como sea posible.

        - Delay allocation retrasa la asignación mientras los datos están en caché, hasta que realmente se escriben en el disco.
    - Persistent pre-allocation:
        - Permite a las aplicaciones reservar espacio en disco antes de necesitarlo realmente.
        - No hay datos en estas áreas preasignadas hasta que la aplicación los escriba en el futuro.



XFS

- Filesystem de 64 bits. Dividido en regiones llamadas “allocation groups”.
- 16 EB tamaño máximo de file system, 8EB tamaño máximo de archivo.
- Red Hat 7.0 lo incluye como su FS default (CentOS desde la versi´on 7.2)
- Uso de extents. Inodos asignados dinámicamente.
- Tiene Journaling siendo el primer FS de la familia UNIX en tenerlo.
- Tiene mayor espacio para atributos extendidos (hasta 64KB).
- Contra: no es posible achicar un FS de este tipo.

ProcFS

- pseudo-filesystem montado comúnmente en el directorio /proc .
- Provee una interface a las estructuras de datos del kernel.

- Presenta información sobre procesos y otra información del sistema en una estructura jerárquica de archivos.
- No existe en disco, el kernel lo crea en memoria. Mayoría de los “files” de solo lectura, aunque algunos pueden ser modificados (/proc/sys).

SysFS

- Con el paso del tiempo, /proc se convirtió en un desorden. Como solución a esto se usa sysfs.
- Es como procfs pero mejor organizado.
- Exporta información sobre varios subsistemas del kernel, dispositivos de hardware y sus controladores, módulos cargados, etc. desde el espacio del kernel hacia el espacio del usuario. También permite la configuración de parámetros.
- SysFS se monta en /sys .

VFS

- También conocido como Virtual Filesystem Switch. Permitía acceso local, UFS, y remoto, NFS, en forma transparente.
- Capa de software en el kernel que provee la interface del file system a los programas en el espacio del usuario.
- Permite la coexistencia de diferentes file systems.
- Permite montar sistemas de archivos remotos.
- Procesos utilizan system calls para acceder al VFS: open, stat, read, write, chmod, etc.
- VFS tiene una estructura compuesta de 4 objetos: superblock, inode, dentry y file.
- Por cada filesystem específico existe un módulo que transforma las características del file system real en las esperadas por el VFS.
- VFS es independiente de los fs implementados.
- Hace de interprete entre varios fs montados.

dumpe2fs

Muestra la configuración y estado del sistema de archivos de la partición deseada. Imprime la información del grupo de superbloques y bloques para el sistema de archivos presente en la partición seleccionada.

Incrementar la cantidad de inodos de un file system

- Crear de un nuevo file system con más inodos:
    - Opción más segura.
        - Se debería realizar una copia de seguridad de los datos, crear un file system con la cantidad deseada de inodos y después restaurar los datos en el nuevo file system.
- Usar herramientas de ajuste avanzado:
    - Existen herramientas avanzadas que permiten ajustar la cantidad de inodos en un file system existente, como resize2fs o tune2fs.
        - Tienen limitaciones y riesgos asociados.
        - Recomendable leer cuidadosamente la documentación y realizar pruebas en un entorno de prueba antes de aplicar cambios en producción.
- Convertir el file system a otro:
    - No aumentará directamente la cantidad de inodos en el file system, convertirlo a otro tipo que permita más inodos podría ser una solución alternativa. Esto también conlleva riesgos:
        - Iniciar la PC desde un Live USB.
        - Utilizar el comando lsblk para listar las particiones.
        - Crear las carpetas /mnt/fuente y /mnt/destino con mkdir.
        - Montar la partición a aumentar en /mnt/fuente y la que almacenará temporalmente una copia de los datos en /mnt/destino con mount.
        - Copiar todos los archivos de /mnt/fuente a /mnt/destino, utilizando cp o rsync.
        - Desmontar /mnt/fuente con umount.
        - Crear una nueva partición con más inodos, reemplazando el sistema de archivos existente con mkfs.

- Montar esta partición en /mnt/fuente con mount.
- Copiar todos los archivos de /mnt/destino a /mnt/fuente, utilizando cp o rsync.
- Reiniciar la PC, arrancando el sistema operativo original.

stat

```
  File: /etc/group
  Size: 879             Blocks: 8          IO Block: 4096   regular file
)evice: 8,1     Inode: 130894      Links: 1
\ccess: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
\ccess: 2024-05-04 13:37:04.339999932 -0300
Modify: 2024-04-01 14:14:57.199999998 -0300
Change: 2024-04-01 14:14:57.267999998 -0300
 Birth: 2024-04-01 14:14:57.199999998 -0300
```

Access – fecha del último acceso.

Modify – la última vez que el archivo fue modificado (el contenido)

Change – la última vez que los metadatos del archivo fueron modificados (ejemplo, los permisos).

Birth – fecha de creación (esto es propio de ext4)

Link simbólico y Hard link

- Link simbólico:
  - Apunta otro archivo o directorio mediante su ruta absoluta o relativa.
  - Si el archivo original se elimina, el enlace simbólico queda roto.
  - Cada enlace simbólico dispone de su propio número de inodo y es diferente al del archivo original.
  - Se pueden crear para directorios.
- Hard-link:
  - Entrada de directorio adicional que apunta al mismo nodo de i-nodo (estructura de datos que almacena información sobre un archivo) que el archivo original.

    - Si se elimina el archivo original, el hard-link sigue apuntando al contenido del archivo.
    - El espacio del disco no se libera hasta que todos los enlaces duros se eliminen.
    - No se puede crear para directorios.

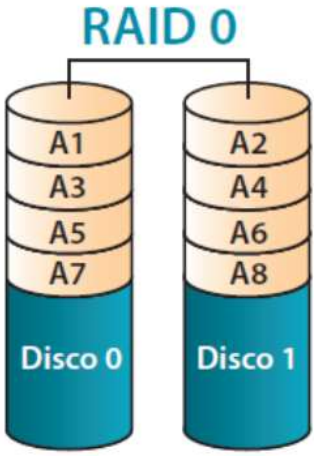
RAID

- Permite usar múltiples discos para que el sistema de discos sea más rápido, grande y confiable.
- Tiene 6 niveles pero solo se usan 0, 1, 5 y 6, y combinaciones de ellos. Las ventajas son mejor performance, mayor capacidad y aumento de confiabilidad.
- Para los file systems es un arreglo lineal de bloques que puede ser leído y escrito. Internamente, debe calcular qué disco/s deben acceder para completar la solicitud.
- La cantidad de accesos físicos de I/O depende del nivel de RAID que se está utilizando.
- Son diseñados para detectar y recuperarse de determinados fallos de discos.
    - Los spare disks son discos disponibles para reemplazo de discos en falla.
- Si un disco falla en el RAID está en modo degradado.

Raid 0

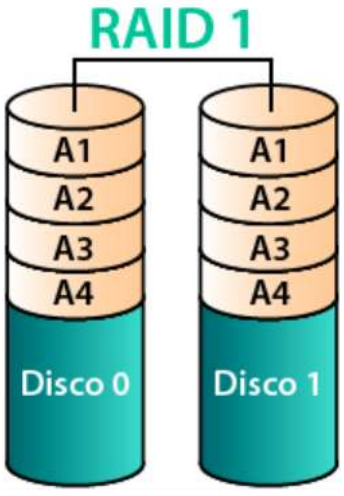
- No es un nivel de RAID en absoluto. No existe redundancia.
- Array de discos con “striping”, proceso de dividir datos en bloques y esparcirlos en muchos dispositivos de almacenamiento, a nivel de bloque.
- Necesita 2 o más discos para conformarse.
- La capacidad del RAID es la sumatoria de la capacidad de los discos participantes.
- Los discos deben ser del mismo tamaño, si no se toma el tamaño del más pequeño.
- Sirve como “upper-bound” en cuanto a performance y capacidad

- Si falla un disco, los datos de todos los discos se vuelven inaccesibles.
- El chunk size indica la cantidad mínima de datos leídos/escritos en cada disco de un array durante una operación de lectura/escritura.



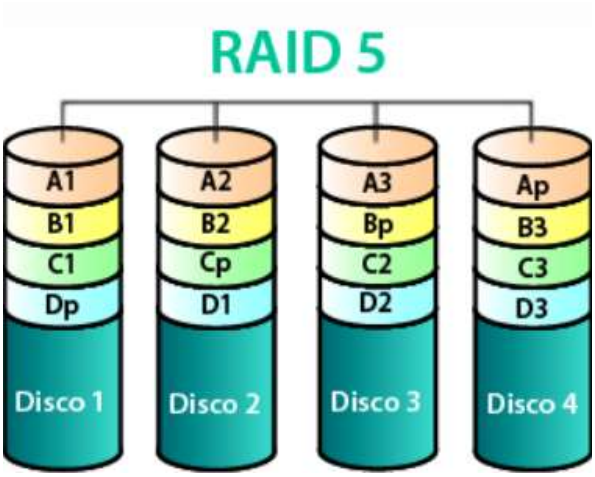
Raid 1

- Asegura redundancia mediante el mirroring (espejado) de datos.
- No hay striping de datos.
- Almacena datos duplicados en discos separados o independientes.
- Mínimo de 2 discos. Trabaja con pares de discos.
- Ineficiente por la escritura en espejo, desperdicia el 50% de la capacidad total.



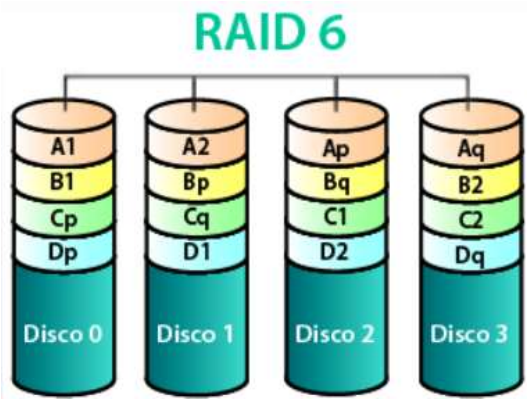
Raid 5

- Striping a nivel de bloque y paridad distribuida. Una de las implementaciones más utilizadas.
- Distribuye la información de paridad entre todos los discos del array.
- Se requieren mínimamente 3 discos.
- Alto rendimiento, sin cuello de botella.
- No ofrece solución al fallo simultáneo de discos.
- Aprovechamiento de n-1 discos.



Raid 6

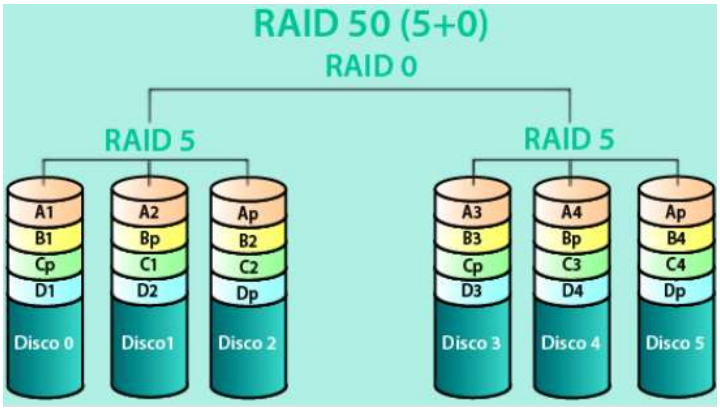
- Striping a nivel de bloque y doble paridad distribuida.
- Recomendado cuando se tienen varios discos.
- Se requieren mínimamente 4 discos.
- Alta tolerancia a fallos (hasta dos discos).
- Operaciones de escritura más lentas debido al cálculo de doble paridad.



Híbridos

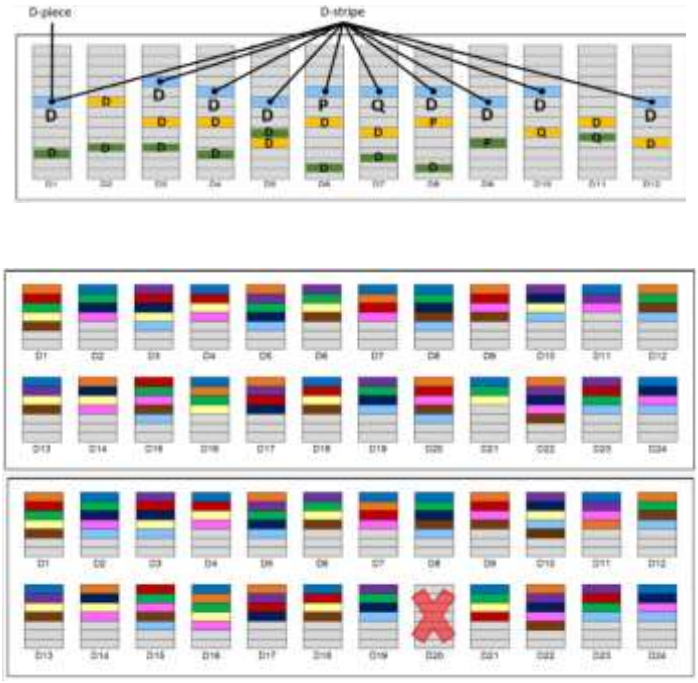
Es posible combinar los distintos niveles de RAID

- RAID 0+1: Mirror of Striped Disks
- RAID 10(1+0): Stripe of Mirrored Disks
- RAID 50(5+0)
- RAID 100
- etc.



Dynamic Disk Pools (DDP)

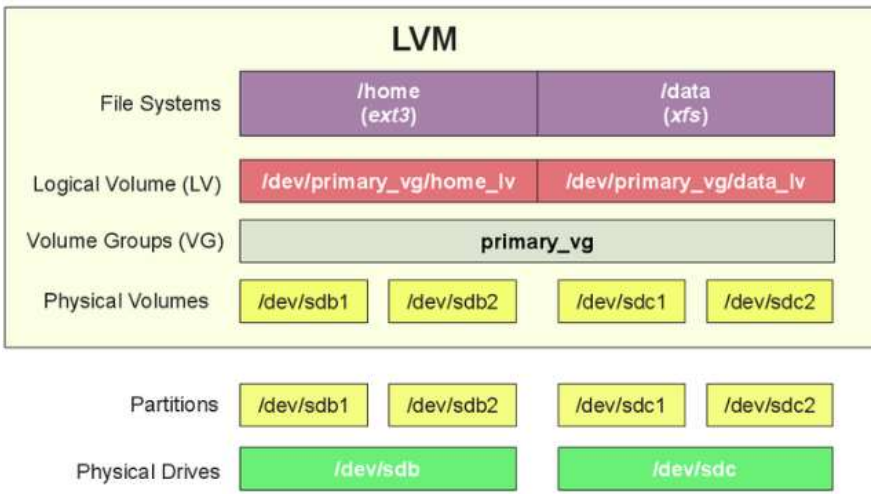
- Como los discos son cada vez más grandes, las técnicas de RAID son lentas. Recuperarse lleva tiempo.
- DDP distribuye dinámicamente los datos, lo que le da capacidad de spare e información de protección a través de todos los discos del pool.
- Necesita 11 discos como mínimo. Puede extenderse hasta cientos de discos.
- Datos se distribuyen en todos los discos del pool sin importar cuantos discos lo componen.
- D-Stripe: distribuidos en 10 discos por un algoritmo optimizado. Cada parte en un disco es un D-Piece. Contenido similar al RAID 6: 8+2
- D-Piece: sección contigua dentro de un disco físico



LVM

- Provee un método más flexible para alocar espacio en los dispositivos de almacenamiento masivo.
- Provee una capa de abstracción entre el almacenamiento físico y el file system. Es una capa de software que se mete entre el SO y el FS.
- Permite la creación de particiones a través de uno o más dispositivos de almacenamiento.
- Los beneficios son el resize online de las particiones, snapshots, mirroring/stripping, etc.
- Las particiones pueden ocupar varios discos.
- Los principales componentes de LVM son
    - Physical Volume (PV): dispositivos físicos o particiones que serán utilizados por LVM.
    - Volume Group (VG): grupo de PVs. Representa el “data storage”.
    - Logical Volume (LV): cada una de las partes en las que se dividen los VGs, equivalentes a una partición.
    - Physical Extent (PE): unidades direccionables en las que se divide cada PV.

- Logical Extent (LE): unidad de alocación básica en los LVs. Puede ser de distinto tamaño al del PE



- Dispositivos de almacenamiento están bajo el control de LVM después de proceso inicialización.
- Dispositivos pueden ser de diferentes tamaños.
- Básicamente, el espacio de todos los volúmenes físicos se agregan para formar un gran volumen de almacenamiento (Volume Group).
    - Es posible generar varios VG dentro de un PV.
- Luego se generan particiones lógica en el VG generado en el paso anterior.
    - Hay que indicarle el tamaño. Dos formas:
        - Trabajando con bytes
            - Se termina calculando de la siguiente manera para respetar el tamaño de los PE dado que a cada volumen logico se le asigna una cantidad de PEs garantizando que se utilice de manera eficiente el espacio asignado en los PEs.
                - $\text{ceil}\left(\frac{\text{NumBytes}}{\text{TamañoPE}}\right) * \text{TamañoPE}$
        - Trabajando con PE
- Luego hay que formatear el LV (con un fs) y montarlo en un punto de montaje.
- Si me quedo sin espacio en una partición, extiendo el logical volumen donde está la misma (asumiendo que hay espacio libre en el VG), por consecuencia para que se reflejen los cambios debo incrementar el tamaño del fs del volumen.

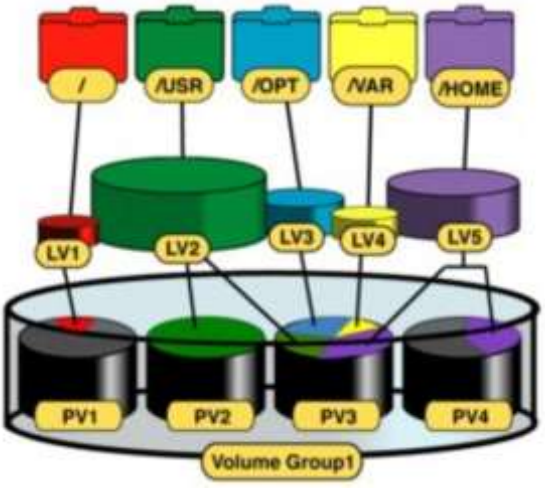
- Si no hay mas espacio en el VG, debería agregar otro volumen físico y extender el tamaño del VG.

Resumen para entender mejor

Un volume group (VG) es una colección de physical volumes (PVs), que crea un grupo de espacio en el disco a partir del cual se pueden asignar logical volumes (LVs).

Dentro de un VG, el espacio en disco disponible para la asignación se divide en unidades de tamaño fijo llamadas extensiones. Una extensión es la unidad de espacio más pequeña que se puede asignar. Dentro de un PV, las extensiones se denominan physical extents.

Un LV se asigna en logical extents del mismo tamaño que las physical extents. Por lo tanto, el tamaño de la extensión es el mismo para todos los LVs lógicos del VG. El VG asigna las logical extents a las physical extents.



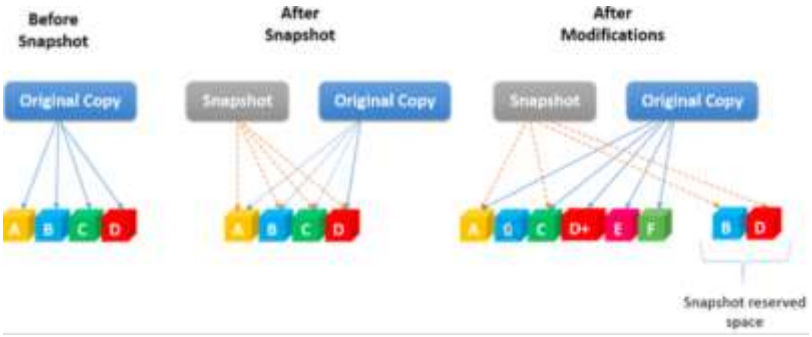
Snapshots

- LVM snapshot copia el punto en el tiempo de un volumen lógico.
- Contiene metadata y los bloques de datos de un LV origen que hayan sido modificados desde que se generó el snapshot.
- Se crean instantáneamente y persisten hasta que se los elimina.
- Snapshot es un Logical Volume dentro del Volume Group.
- Puede ser solo lectura o lectura y escritura.

- Usa C-O-W.

C-O-W

- Provee consistencia en los datos y metadatos y snapshots instantáneas.
- C-O-W escribe los datos en una nueva ubicación, luego se actualizan los metadatos.
- Cuando se realiza una snapshot LVM, al crearse el snapshot solo se copian los metadatos. Recién cuando haya una modificación (en la primera nada más, copy-on-firts-write) de datos los bloques de datos se copian al snapshot (a un volumen lógico temporal) antes de ser modificados en el volumen original y ahora la snapshot apuntara a esa copia. Los bloques que no se modificaron, no estarán en la snapshot. Para reestablecer la snapshot, se copian los bloques de datos a la ubicación original. Si se desea mantener los cambios realizados, entonces se borra la snapshot.



BTRFS

- BTRFS:
  - SO basado en COW
  - Tamaño máximo de volumen: 16EiB.
  - Tamaño máximo de archivo: 16EiB
  - Cantidad máxima de archivos: 2^{64}
  - Trabaja con:
    - Extents.
    - C-O-W.

- Por default, COW es usada en todas las escrituras al file system (puede ser deshabilitada). Los nuevos datos se crean como siempre mientras que cuando se modifica un dato se copian en un nuevo espacio (no se sobreescriben) y luego se modifican los metadatos; no es necesario journaling.
    - cp -relinks o btrfs subvolume no duplica datos.
- Proporciona ciertos niveles de atomicidad y consistencia en las operaciones de escritura, lo que garantiza que las operaciones se completen de manera segura y sin corrupción de datos.

- Alocación dinámica de inodos.
- Soporte integrado de múltiples dispositivos:
    - RAID0 (stripe) similar al RAID0 tradicional.
    - RAID1 (mirror) similar al RAID1 tradicional.
    - RAID5 y RAID6 similares a los RAID1 y RAID6 tradicionales.
- Checksum de datos y metadatos (con CRC329 y Scrub.
- Subvolumenes.
- Snapshots.
- Es posible la reduplicación, pero no se realiza online. Son herramientas adicionales.
- Por defecto BTRFS replica los metadatos (DUP) pero no los datos (single).

- Permite:
    - Agregar o remover dispositivo de bloques.
    - Agrandar o achicar volúmenes.
    - Desfragmentación online.
    - Compresión (zlib, LZO, etc.).

- ZFS
    - Tamaño máximo de volumen: 256 ZiB.
    - Tamaño máximo de archivo: 16EiB
    - Cantidad máxima de archivos: 2^{48}

- Trabaja con:
  - C-O-W.
  - Self-healing.
  - Soporte integrado de múltiples dispositivos
    - RAIDZ.
    - RAIDZ2 y RAIDZ3.
    - Mirrored vdevs.
  - Checksum de datos y metadatos y Scrub.
  - Subvolumenes (datasets)
  - Snapshots.
- Permite :
  - Gestión avanzada del almacenamiento,
    - Agrupación de dispositivos (vdevs)
    - Caché de lectura y escritura (L2ARC)
    - Caché adaptativa (ARC).
  - Compresión.
- No cuenta con desfragmentación.

Subvolúmenes

- No es necesario crear particiones, se crea un Storage Pool en el cual se crean subvolúmenes.
- Un subvolumen puede ser accedido como un directorio más o puede ser montado como si fuese un dispositivo de bloques, incluso aunque no lo sea.
- Cada subvolumen puede ser montado en forma independiente y pueden ser anidados.
- No pueden ser formateados con un filesystem diferente.
- Existe un subvolumen en cada filesystem BTRFS llamado top-level subvolume.
- Es posible limitar la capacidad de cada subvolumen (qgroup / quota).
- No pueden extenderse a otro BTRFS filesystem

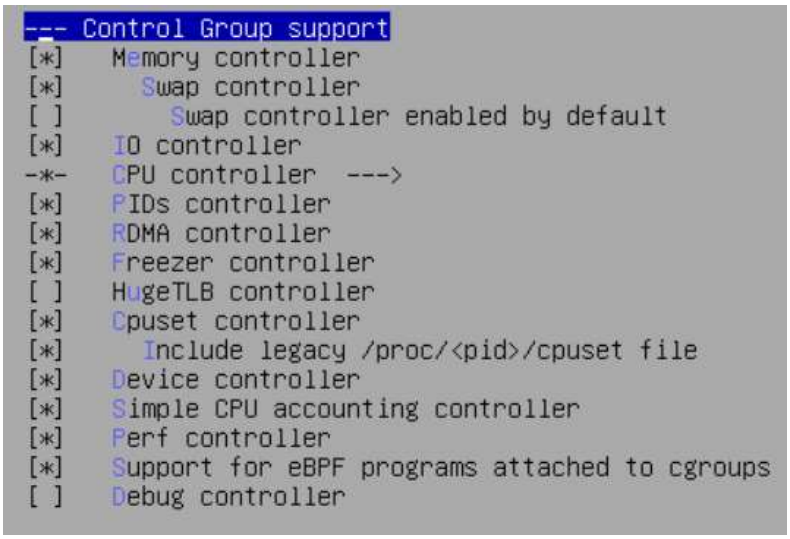
Practica 4

Service isolation - chroot

- Es una forma de aislar aplicaciones del resto del sistema.
- Cambia el directorio raíz de un proceso, afectando sólo a ese proceso y a sus hijos.
- Al entorno se llama “jail chroot”.
- No puede acceder a archivos y comandos fuera de ese directorio.

control groups

- El kernel no puede determinar qué proceso es importante y cuál no. Todos tienen el mismo trato.
- Control groups permite agrupar de forma jerarquica procesos y así limitar y monitorear uso de recursos.
- La interfaz que provee el kernel funciona mediante un pseudo-filesystem llamado cgroups.
    - Proporciona control de grano fino en alocación, priorización, denegación y monitoreo de recursos.
    - Limita recursos (resource limiting), prioriza grupos (prioritization), medición de uso de recursos (accounting) y permite controlar la vida de los procesos (control).
- Los procesos son ajenos a cgroups.
- No son volátiles los valores que se le dan a un cgroup; para persistirlos, se debe decirle al SO que guarde al apagar y cargue al iniciar la PC
- Actualmente hay 12 subsistemas definidos



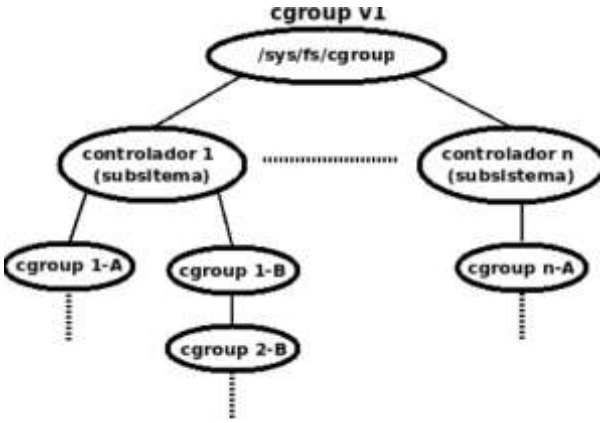
- Hay dos versiones:
    - Ambas versiones pueden ser montados en el mismo sistema
    - Una jerarquía de un controlador no puede estar en ambos cgroups simultáneamente
        - Una jerarquía, por ejemplo, es el control de la memoria.

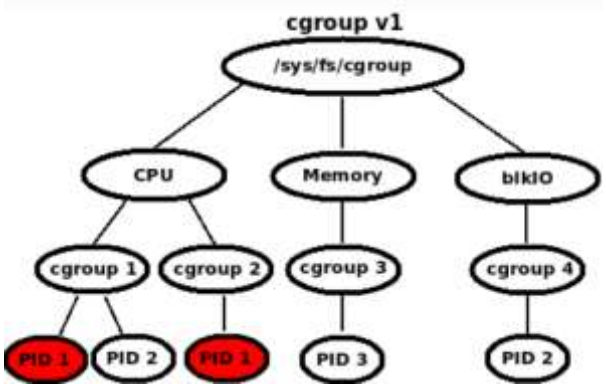
cgroups v1

- Asocia un conjunto de tareas con un conjunto de límites para uno o más subsistemas (o controladores), que son componentes que manejan el comportamiento de los procesos de ese cgroup, cada uno representa un recurso.
- La jerarquía es un conjunto de cgroups organizados en un árbol. Por cada una, la estructura de directorios refleja dependencia de cgroups.
    - mkdir /sys/fs/cgroup/cpu/"nombre_cgroup"
        - Crear un cgroups dentro del subsistema cpu
        - Se puede utilizar cgcreate
- Un proceso puede pertenecer a muchas jerarquías; pero dentro de cada jerarquía, sólo a un cgroup.
- Cada jerarquía es definida mediante la creación (mkdir), eliminación (rm), renombrado (mv) de subdirectorios dentro del pseudo-filesystem.
- Cada cgroup se representa por un directorio en una relacion padre-hijo. Los directorios tienen archivos que pueden ser escritos o leídos.
    - Por ejemplo: /sys/fs/cgroup/cpu/procesos/proceso1

        - Dentro de los CPUs, tengo procesos, y a ese proceso le doy acceso o no a cosas, etc.
    - Proceso creado con ”fork” pertenece al mismo cgroup que el padre.
- En cada nivel de la jerarquía se pueden definir atributos (limites), que no pueden ser excedidos por los cgroups hijos.
- Si queremos que un proceso pertenezca a la jerarquía, hay que agregarlo a cgroup.props.
- Una vez que se definen los grupos se les agregan ID a los procesos. Se pueden asignar threads de un proceso a diferentes grupos.
- Para montar uno o varios controladores se utiliza el comando:
    - mount -t cgroup -o <controlador> none /sys/fs/cgroup/<controlador> para montar un controlador en particular
    - mount -t cgroup -o all cgroup /sys/fs/cgroup para montar todo los controladores
    - Luego el Kernel crea los archivos y carpetas necesarios
        - Según la jerarquía, va a crear distintos archivos que no necesariamente serán los mismos
        - Por ejemplo, cpuset/ es para decir que procesadores queremos que usen los procesos que uno va a poner dentro de ese cgroup.
        - Con el echo y redirección al archivo cpuset/<grupo> /cpuset.cpus, le digo que esos procesos usen los núcleos 0 a 2 y 4.
        - También para cpuset/my_group/cpuset.procs, puedo especificar la lista de PIDs que estarán dentro.
        - Los límites por defecto son los máximos, y uno después lo limita.
- Controladores pueden ser montados en pseudo-filesystems individuales o en un mismo pseudo-filesystem.
- Un controlador puede ser desmontado si no está ocupado: no tiene cgroups hijos (umount /sys/fs/cgroup/<controlador>)
    - Si se hace unmount el SO elimina los archivos y carpetas que se crearon cuando se montó.

- Cada cgroup filesystem contiene un único cgroup raíz al cual pertenecen todos los procesos.





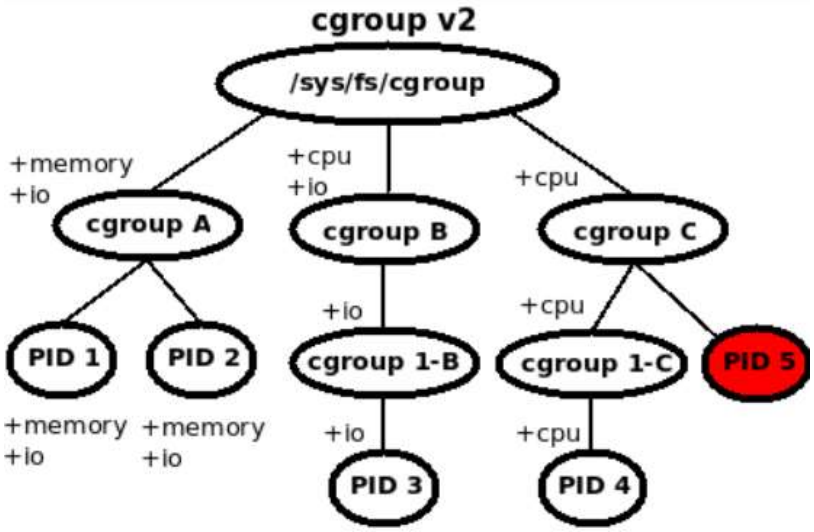
- El proceso con PID 1 no puede estar en dos cgroups dentro de la misma jerarquía (en CPU).

cgroups v2

- Es el reemplazo de v1.
- Todos los controladores son montados en una jerarquía unificada.
- No se permiten procesos internos, excepto en cgroup root. Procesos deben asignarse a cgroups sin hijos (hojas).
- No se puede especificar un controlador en especial para montar.
    - mount -t cgroup2 none /mnt/cgroup2 para montar todo
- Todos los controladores son equivalentes a los de v1, menos net_cls y net_prio
- Cada cgroup en la jerarquía v2 contiene, entre otros los siguientes archivos:

    - cgroup.controllers: Archivos de solo lectura que indica los controladores disponibles en este cgroup
    - cgroup.subtree_control: Lista de controladores habilitados en el cgroup
        - Creo las inferiores, las creará con todos los subsistemas habilitados, pero con el cgroup.subtree_control, le indico cuales son los que quiero habilitar
        - Con el +nombre indico que está habilitado, con el -nombre, deshabilitado.
            - echo '+pids -memory' > cgroup.subtree control

- Inicialmente, solo el cgroup root existe y todos los procesos pertenecen a el
- El mkdir CG_NAME y rmdir CG_NAME para agregar/eliminar cgroups.
- Está el archivo cgroup.props de lectura/escritura, para decirle qué procesos tienen ese recurso
- Ya no se tienen jerarquías preestablecidas para memoria, E/S, etc., si que se tiene la jerarquía unificada, se crean cgroups, y en cada uno habilito o deshabilito controladores
- Los hijos heredan lo que se habilita para un nivel superior
- Si se trabaja sobre /sys/fs/cgroup/unified/ se está trabajando sobre v2.



- El proceso con PID 5 no puede estar en un cgroup con hijos.

Namespace Isolation

- Permite abstraer un recurso global del sistema para que los procesos dentro de ese namespace piensen que tienen una instancia aislada de ese recurso.
- Limitan lo que un proceso ve y por consecuencia lo que usa.
- Toda modificación de ese recurso queda dentro del namespace.
- Un proceso puede estar en un namespace de un tipo a la vez.
- Cuando el proceso abandona o finaliza se elimina el namespace.
- Un proceso puede usar alguno, todos o ningun namespace del padre.
- Linux provee namespaces:
    - IPC
    - Red
    - Montaje
    - PID
    - Usuarios
    - UTS
    - Cgrous
    - Time
- Flag: usado para indicar el namespace en las system calls.
- Se usan nuevas systemcalls:
    - clone() : similar al fork. Crea un nuevo proceso y lo agrega al nuevo namespace especificado.
    - unshare() : agrega el proceso actual a un nuevo namespace. Es similar a clone, pero opera en el proceso llamante. Crea al nuevo namespace y hace miembro de él al proceso llamador.
    - setns() : agrega el proceso actual a un namespace existente. Desasocia al proceso llamante de una instancia de un tipo de namespace y lo reasocia con otra instancia del mismo tipo de namespace.
- Cada proceso tiene un subdirectorio que contiene los namespaces a los que está asociado: /proc/[pid]/ns .
- PID permite tener muchos arboles de procesos anidados y aislados.
    - El mismo proceso tiene dos PIDs (o más):
        - Dentro del host anfitrión

        - Dentro del namespace
    - Un mismo PID puede repetirse entre namespaces (pero no dentro de un mismo namespace)
- Mount permite aislar tabla de montajes.

Virtualización vs Emulación

Virtualización: particionar un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador.

- Los SO guest deben ejecutar la misma arquitectura de hardware sobre la que corren.
- No requiere que los guest se modifiquen.
- El VMM analiza el flujo de ejecución.
- Los bloques que contienen instrucciones sensibles son modificados.
- Los bloques con instrucciones inocuas se ejecutan directamente en el hardware.

Emulación: provee toda la funcionalidad del procesador deseado a través de software.

- Se reescribe el conjunto completo de instrucciones.
- Se puede emular un procesador sobre otro tipo de procesador.
- Lenta.

Hipervisor

Componente de software que administra varias máquinas virtuales en una computadora. Separa las “aplicaciones/SO” del hardware subyacente. Garantiza que cada máquina virtual reciba los recursos asignados y no interfiera con el funcionamiento de otras máquinas virtuales.

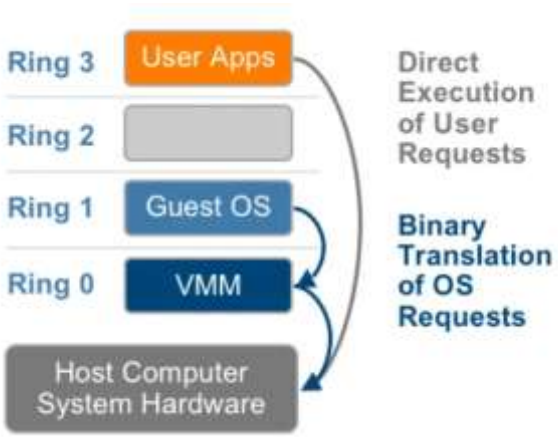
Clasificación:

- Hipervisores de tipo 1: aparecen dos clasificaciones:

    - Software dedicado que fue creado exclusivamente para ofrecer ambientes de virtualización o VMM.
    - SO de propósito general como Windows o Linux que ofrecen capacidades de virtualización, es decir son VMM, y además un clásico SO.
- Hipervisores de tipo 2: también llamados hosted hypervisors, son aplicaciones que corren sobre un SO estándar y confían en el SO preexistente para manejar los recursos virtuales.

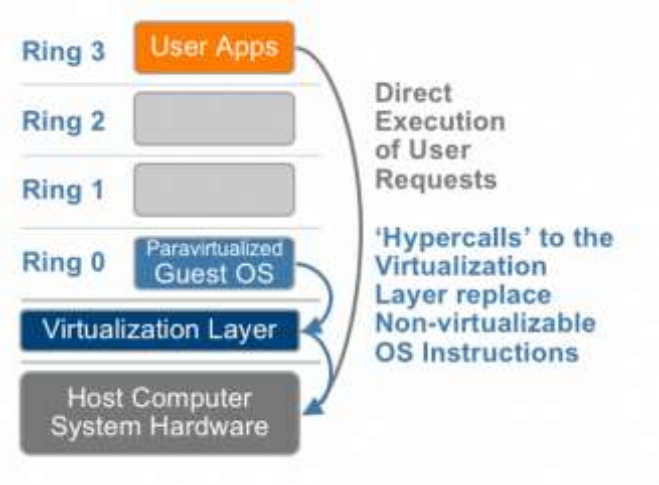
Tipos de virtualización

- Virtualización completa mediante traducción binaria
    - No necesita asistencia de hardware ni del SO.
    - El hipervisor simula completamente el hardware subyacente.
    - Usa una combinación de ejecución directa y traducción binaria permitiendo la ejecución directa de instrucciones de CPU no sensibles, mientras que las instrucciones de CPU sensibles se traducen sobre la marcha.
        - Para mejorar el rendimiento, el hipervisor mantiene una caché de las instrucciones recientemente traducidas.
        - Ejecución directa del código a nivel de usuario en el procesador para alto rendimiento.

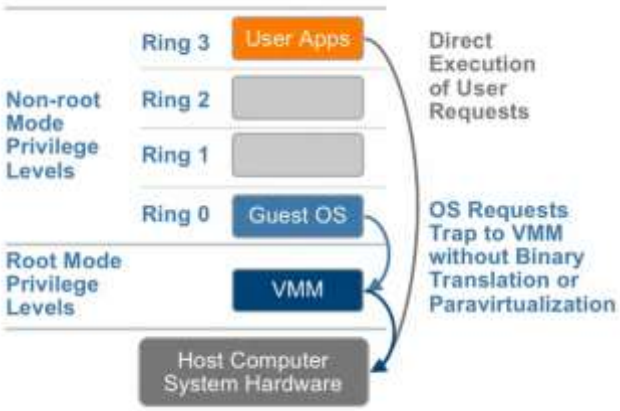


- Virtualización o paravirtualización asistida por SO

- Modificación del kernel del SO para reemplazar instrucciones no virtualizables con hiperllamadas directas al hipervisor.
- El hipervisor no simula el hardware subyacente, si no que proporciona hiperllamadas. El SO guest usa las hiperllamadas para ejecutar instrucciones sensibles de la CPU.
- Menos portátil.
- Mejor rendimiento.



- Virtualización asistida por hardware (primera generación)
  - El hardware subyacente proporciona instrucciones de virtualización de CPU para ayudar a la virtualización.

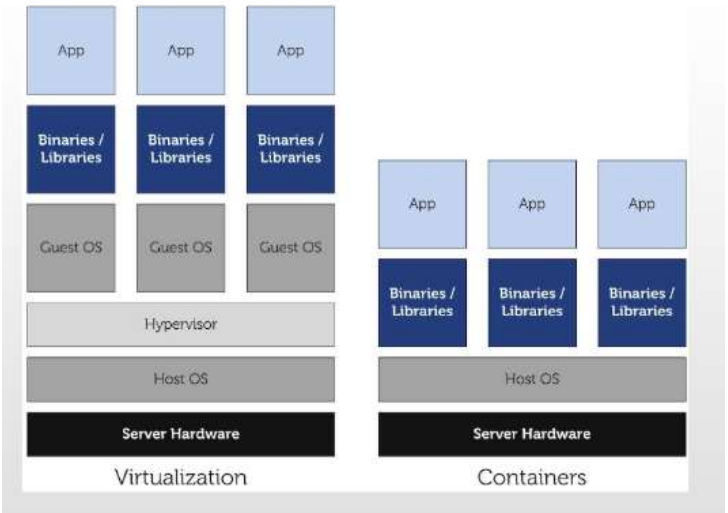


Binary translation y trap-and-emulate

- Binary translation: traducción dinámica de las instrucciones del SO guest en virtual kernel mode que no pueden ejecutarse directamente en el hardware del host.
- Trap-and-emulate cuando el SO guest ejecuta una instrucción privilegiada se genera un trap que transfiere el control al hipervisor que emula la instrucción privilegiada.

Containers

- Tecnologia liviana que permite ejecutar múltiples sistemas aislados en un único host.
- Las instancias se ejecutan en espacio del usuario y comparten el mismo kernel.
- Dentro de cada instancia son como máquinas virtuales. Por fuera, son procesos normales del SO.
- Método de virtualización más eficiente: mejor performance, booteo más rápido.
- No es necesario un software de virtualización tipo hipervisor.
- No es posible ejecutar instancias de SO con kernel diferente al SO base



Docker

- Docker permite ejecutar una aplicacion en containers aislados. Los containers son livianos y tienen todo lo que necesita la app para funcionar.
- Está divido en 3 componentes: demonio dockerd, API rest y la CLI docker.
- Utiliza arquitectura cliente servidor, pueden ejecutarse en el mismo sistema o diferentes y se comunican por API rest.
- dockerd escucha por la api request y administra los objetos de docker.
- Las herramientas que utiliza docker son:
    - Namespaces: Docker lo utiliza para proveer el espacio de trabajo aislado que denominamos container. Cada container su propio namespace.
    - Control groups: Para, opcionalmente, limitar los recursos asignados a un contenedor.
    - Union file systems: Se utilizan como filesystem de los containers. Es un mecanismo de montajes que permite que varios directorios tengan el mismo punto de montaje apareciendo como único file system.
- Las imagenes de docker son paquetes de lectura que tienen todo lo necesario para ejecutar aplicaciones, puede basarse en otras. Son una coleccion de archivos.
- Un registry es un almacen de imagenes de Dockers, por defecto usa Docker Hub.
- Un container es una instancia de la imagen.
- Un dockerfile es un archivo que define como construir una imagen.
- Las imágenes tienen capas que se montan una sobre otra, solo la última es R/W. Cada capa es una instrucción en el dockerfile de la imagen. Cada capa es diferente a la otra.
- La diferencia entre un container y una imagen es la capa escribible.
- Los archivos creados dentro de un contenedor se almacenan en una capa escribible, los datos no son persistentes.
- Escribir dentro del contenedor requiere un driver del union filesystem.
- Docker tiene dos opciones para almacenar datos en el host y que sea persistente:

    - Volúmenes: se almacenan en una parte del filesystem administrada por Docker. Método recomendado
    - Bind mounts: está en cualquier parte del filesystem y puede ser modificada por procesos que no son de Docker.
- Docker usa la subred predeterminada 172.17. 0.0/16 para los contenedores.
    - El demonio de Docker se encarga de crear subredes dinámicas y asignar direcciones IP a los contenedores.
    - Las direcciones IP se asignan mediante un servidor DHCP interno administrado por Docker.

Comandos

- docker pull imagen: descarga la imagen.
- docker run imagen: ejecuta la imagen.
    - Con -it indica a Docker que inicie el contenedor en modo interactivo con una terminal.
- docker image build -t NOMBRE: crea una imagen a partir de un Dockerfile.
- docker push NOMBRE usuario/repositorio: sube la imagen al Docker Hub pero antes hay que hacer Docker Login
- docker info: info general
- docker ps: containers en ejecución
- docker image/containers ls: lista imagenes o containers
- docker commit CONTAINER REPOSITORY:TAG: commitea los cambios del container a la imagen

Practica 5

Docker Compose

- Los sistemas están formados por varias partes que necesitan comunicarse. Docker-compose es una herramienta que facilita hacer el despliegue de aplicaciones compuestas en múltiples contenedores.

- Se define a Docker Compose como el conjunto de la herramienta (el binario) y los archivos de configuración llamados “compose file” que tiene los recursos que se necesitan.
- Actualmente Compose es un plugin de Docker que se invoca para interpretar los “compose file” y desplegar los contenedores allí definidos.
- Un compose file está escrito en YAML y tiene las características de los contenedores a desplegar, tiene toda la configuración.
    - Se lo llama: docker-compose.yaml.
    - A cada contenedor definido en este archivo se lo denomina “service”
        - Un service:
            - Define uno o más containers con la misma imagen y configuración.
            - Se conecta con otros services a través de “networks”.
            - Almacenan datos persistentes en “volumes”.
            - Por default name = domain name
- Cuando se invoca el binario docker-compose, este busca en la carpeta desde donde es invocado algún archivo con nombre docker-compose.yml .
    - También se le puede pasar como parámetro cuál es el archivo compose a utilizar.
- En base al contenido del compose se iniciarán los distintos contenedores con las características definidas para cada uno.
- Ventajas: Docker compose es simple (todo definido en un archivo con estructura determinada y documentada), replicable (se quiere desplegar nuevamente el servicio, corro el mismo archivo), compartible (para compartir solo se tiene que proveer el compose file) y versionable (es fácil usar control de versiones).
- No se necesita Dockerfile porque se están utilizando imágenes de contenedores predefinidas disponibles en Docker Hub (con Dockerfiles ya definidos), haciendo que no sea necesaria la definición de Dockerfiles personalizados.
- Los contenedores se comunican entre si usando el nombre del servicio como nombre de dominio y Docker Compose se encarga de la resolución de nombres.

Versiones

Existen tres versiones para el formato de archivo de docker-compose.yml:

- Versión 1:
  - Está obsoleta.
  - No requería especificar una versión en el archivo docker-compose.yml.
  - Características básicas para definir servicios, volúmenes y redes.
  - Todos los servicios se declaran en la raíz del documento
  - No permite declarar volúmenes nombrados, redes o argumentos de construcción
  - Todos los contenedores se conectan a la misma red predeterminada, de tipo Bridge
- Versión 2.x
  - Introducción de la sección version.
  - Mejoras en la configuración de redes y volúmenes.
  - Todos los servicios se declaran dentro de la clave services
  - Permite declarar volúmenes nombrados, dentro de la clave volumes
  - Permite declarar redes, dentro de la clave networks
  - Por default, los contenedores se conectan a una misma red, y se usa como nombre de host, el nombre de servicio
  - Las versiones 2.1 a 2.4 agregan otras claves y características.
  - Soporte para dependencias de servicios mediante depends_on.
  - Añadido soporte para configuraciones más avanzadas como healthcheck, deploy, y secrets.
  - Permite el uso de comandos de construcción (build) más avanzados.
- Versión 3.x:
  - Ultima versión y la recomendada por Docker.
  - Enfocada en el uso con Docker Swarm, permitiendo la configuración de despliegues en un entorno de clúster.
  - Introducción de la sección deploy para especificar políticas de despliegue (número de réplicas, restricciones de recursos, etc.).
  - Soporte mejorado para secretos y configuraciones (configs).
  - Opciones avanzadas para redes y volúmenes, como configuraciones de driver y driver_opts.

- healthcheck avanzado para verificar el estado de los servicios.
- Soporte para configs y secrets para gestionar configuraciones sensibles y secretas.
- Se remueven y se incorporan algunas claves.

Las versiones 2.x y 3.x comparten estructura pero no algunas opciones. Esto hace que haya una retrocompatibilidad hasta cierto punto dado que no todas las características entre versiones son compatibles.

Estructura

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

- services: define los servicios que componen la aplicación. Cada uno corresponde a un contenedor.
- build: especifica cómo construir una imagen de Docker a partir de un Dockerfile y un contexto de construcción. Puede contener la ruta a un directorio con un archivo Dockerfile, o puede ser un objeto que incluya el contexto de construcción y el Dockerfile.
- image: especifica qué imagen de Docker usar como base para el contenedor.
- volumes: permite definir volúmenes para conservar datos o compartir datos entre contenedores. Se pueden especificar configuraciones de volumen, como rutas de origen, rutas de destino dentro de contenedores y configuraciones de solo lectura.

- restart: permite especificar políticas de reinicio para los servicios, definiendo cómo deben comportarse en caso de fallas o al reiniciar. Las opciones incluyen no, always, on-failurey unless-stopped.
- depends_on: permite definir dependencias entre servicios. Esto garantiza que un servicio se inicie sólo después de que los servicios de los cuales depende estén en funcionamiento.
- environment: permite definir variables de entorno para cada servicio. Sirve para pasar ajustes de configuración a sus contenedores. Cada clave y valor se mapea a una variable de ese nombre y ese valor en el entorno del contenedor.
- ports: se utiliza para mapear puertos del contenedor a puertos en el host. También se puede definir el protocolo (TCP o UDP) para cada puerto.
- expose: expone puertos del contenedor solo a otros contenedores conectados en la misma red, sin exponerlos al host.
- networks: permite definir redes personalizadas para los servicios. Estas permiten que los contenedores se comuniquen entre sí a través de redes aisladas. Se pueden especificar configuraciones de red como alias, direcciones IP y conectividad externa. Hay un networks raíz y un networks en cada servicio particular. En el networks raíz se definen todas las redes que se van a crear.
- healthcheck: permite definir configuraciones de comprobación de estado para los servicios. Los health checks monitorean el estado de los contenedores y pueden usarse para determinar si un servicio está en buen estado o no.

Comandos

- docker compose create y docker compose up: el primero crear todos los contenedores. El segundo lo mismo y además los inicia.
- docker compose stop y docker compose down: el primero frena todos los contenedores. El segundo lo mismo y además los elimina. También elimina las redes y volúmenes asociados pero no las imágenes. Para eliminar volúmenes con nombre, se utiliza la marca --volumes o -v.
- docker compose run y docker compose exec: el primero crea e inicia un nuevo contenedor para ejecutar un comando específico y termina. El segundo

ejecuta un comando dentro de un contenedor en ejecución. El contenedor sigue en ejecución después de que se complete el comando.

- docker compose ps: lista los contenedores gestionados por Docker Compose, mostrando el estado, nombres y puertos mapeados.
- docker compose logs: muestra los logs de salida de los contenedores. Muestra el output de los contenedores.

Tipo de volúmenes

Anonymous volumes: no tienen un nombre definido por el usuario. Docker los crea automáticamente cuando se crea un contenedor y asigna un ID único al volumen. Difícil gestionar.

```
services:
      example_service:
            volumes:
                  - /container/path
```

Docker host-mounted volumes: permite montar un directorio o archivo del host en el contenedor. Puede ser útil para compartir datos entre el host y el contenedor.

```
services:
      example_service:
             volumes:
                  - /host/path:/container/path
```

Named Volumes: tienen un nombre definido por el usuario. Fáciles de gestionar. Docker crea y gestiona volúmenes nombrados y almacena sus datos en una ubicación específica en el sistema host. Se pueden definir como internal (predeterminados) o external.

```
services:
      example_service:
            volumes:
                  - named_volume_name:/container/path
```

Internal: tienen el alcance de un único archivo Docker-compose y Docker los crea si no existen.

```
volumes:
```

```
      named_volume:
services:
      example_service:
            volumes:
                  - named_volume:/container/path
```

External: se pueden usar en toda la instalación de Docker y deben ser creados por el usuario (de lo contrario, falla) usando el comando Docker Volume Create. Hay que definir volumen.

```
volumes:
      named_volume:
            external:true
services:
      example_service:
            volumes:
                  - named_volume:/container/path
```