Resumen parcial

Type	
Materials	
Reviewed	☐

Práctica 1

Introducción

El kernel de Linux es un programa que ejecuta programas y gestiona dispositivos de hardware. Es el intermediario entre el hardware y el software. Es como el sistema operativo. Administra memoria y el uso de CPU. Da soporte a arquitecturas.

Las aplicaciones se comunican con el kernel mediante system calls.

Arquitectura

La arquitectura del kernel Linux se caracteriza por su flexibilidad, escalabilidad y capacidad de personalización.

El kernel de Linux es monolítico híbrido. Los drivers y el código del kernel se ejecutan en modo privilegiado pero se pueden cargar y descargar funcionalidades con módulos.

El código del kernel es muy portable lo que significa que puede ejecutarse en diferentes plataformas sin modificaciones importantes.

El kernel cuenta con gestión de procesos, de memoria y de I/O.

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 muchas cosas:
    - scheduling que disminuye el consumo energético en smartphones
    - soporte de namespaces para android
    - soporte de archivos swap en BTRFS
    - lockdown modo que previene el acceso de procesos de usuario a la memoria del kernel
    - soporte a USB 4
    - nuevo mecanismo para manejar syscalls
    - permite restringir acciones que un conjunto de programas pueden ejecutar en un filesystem
    - soporte inicial para IPv6 (5.19)
- 6.0. Soporta módulos escritos en Rust.

▼ Versionado

Se definen de tres maneras:

- Versión < 2.6: Venían 3 números separados por puntos: `X.Y.Z`. Dos versiones del kernel: números Y `pares` indicaban producción (estable) y los Y `impares` indicaban una versión en desarrollo.
    - X: serie principal. Funcionalidades importantes.
    - Y: indicaba si era de producción o desarrollo.
    - Z: bugfixes.

- Versión ≥ 2.6 y < 3.0: Venían 4 números separados por puntos. `A.B.C.[D]`
  - A: Denota versión.
  - B: Cambios importantes.
  - C: Denota revisión menor. Solo con nuevos drivers o características.
  - D: Se soluciona un error sin agregar funcionalidad.
- Versión ≥ 3.0: Se vuelve al esquema de 3 números.
  - A: Revisión mayor.
  - B: Revisión menor.
  - C: Numero de revisión.
  - rcX: Versiones de prueba.

▼ Compilación

El kernel se compila para: actualizar versión, agregar elementos o funcionalidades, corregir errores, mejorar el rendimiento!!!

Para compilar necesito:

- `gcc`: compilador de C.
- `make`: ejecuta las directivas definidas en los Makefiles.
- `binutils`: assembler, linker.
- `libc6`: archivos de encabezados y bibliotecas de desarrollo
- `ncurses`: permite tener una interfaz con diálogos
- `initrd-tools`: herramientas para crear discos RAM

El proceso es:

1. Descargar código fuente:

```
$ cd /usr/src
$ sudo wget https://kernel.org/pub/linux/kernel/v5.x/linux-5.16.tar.xz
```

2. Preparar el árbol de archivos del código fuente: El código fuente se guarda en `/usr/src`. Si no hay permisos se hace sobre el $home del usuario actual. Se hace una carpeta llamada kernel y se descomprime

ahí lo que se descargo. Generalmente se crea un enlace simbólico llamado Linux apuntando al directorio del código fuente que actualmente se está configurando. `ln -s /usr/src/linux-5.16 /usr/src/linux`.

3. Configurar kernel: El kernel Linux se configura mediante el archivo .**config**. Este archivo, que reside en el directorio de código fuente del kernel, tiene las instrucciones de qué es lo que el kernel debe compilar. Existen tres interfaces que permiten generar este archivo:
   - **make config**: modo texto y secuencial. Tedioso.
   - **make xconfig**: interfaz gráfica utilizando un sistema de ventanas. No todos los sistemas tienen instalado X.
   - **make menuconfig**: este modo utiliza **ncurses**, una librería que permite generar una interfaz con paneles desde la terminal. Generalmente el más utilizado.
4. Construir el kernel a partir del código fuente e instalar módulos.

> Un módulo otorga funcionalidad al kernel sin reiniciar el sistema, lo hace más modular. Se ejecuta en modo kernel, si hay un error puede parar todo el SO. Los módulos disponibles se ubican en `/lib/modules/version` del kernel. Con el comando `lsmod` es posible ver qué módulos están cargados.
>
> El soporte de los módulos puede ser built-in (crece el kernel, se usa más memoria, incrementa el tiempo de arranque, es más eficiente la utilización) o módulo (se carga el modulo y no hay que recompilar, solo se modifica el modulo y no todo el kernel, se cargan bajo demanda).

> 🔧 Patch es un mecanismo que me permite actualizar sobre una versión base. Se compone de archivos `diff`, que indican qué se agrega y qué se quita.
>
> En el kernel hay dos tipos: no incrementales e incrementales.
>
> Es más fácil descargar el diff que descargar todo el código de la nueva versión.
>
> ```
> cd linux; zcat ../patch-5.16.16.gz | patch -p1
> ```

El comando `make` busca el archivo **Makefile**, interpreta sus directivas y compila el kernel. El proceso puede durar mucho tiempo dependiendo del procesador que tengamos.

```
make -jX # X es el número de threads
```

El comando `make modules` compila todos los módulos necesarios para satisfacer las opciones que hayan sido seleccionadas como módulo. Generalmente se encuentra incluida en el comando **make**.

Con `make install` se instala todo. Con `make modules install` se instalan los módulos.

5. Reubicar el kernel.
6. Crear **initramfs**. Este es un sisterma de archivos temporales que se monta en el arranque del sistema. Tiene ejecutables, drivers y módulos. Se desmonta cuando termina el arranque. Se usa el comando `mkinitramfs`.
7. Configurar y ejecutar el gestor de arranque (LILO, GRUB, etc.): para que el gestor de arranque reconozca el kernel hay que ejecutar `update-grub2`. Esto es debido a que el gestor de arranque es quien carga el código del kernel y no sabe cuando se cambia de versión o hay un nuevo kernel, entonces hay que actualizarlo.
8. Reiniciar y probar.
    a. Verificar que el kernel esté instalado en `/boot`.

b. Verificar que los módulos estén instalados en /lib/modules

c. Verificar que el gestor de arranque haya indexado el kernel

d. Paso final: reiniciar y probar

Comandos importantes para la compilación del kernel:

- `make menuconfig`: permite genera el archivo `boot`. Este modo utiliza **ncurses**, una librería que permite generar una interfaz con paneles desde la terminal.
- `make clean`: se utiliza para eliminar todos los archivos que se crearon durante la compilación del kernel y que ya no son necesarios. Esto ayuda a limpiar el directorio de trabajo y a reducir la cantidad de espacio de almacenamiento utilizado.
- `make`: es utilizado en el proceso de compilación de programas y del kernel de Linux. La opción `-j` se utiliza para indicar al comando `make` que realice la compilación en paralelo, utilizando el número de hilos especificado como argumento después de la opción `-j`. Es importante tener en cuenta que el uso de la opción `-j` puede hacer que la salida del comando `make` sea más difícil de seguir, ya que las salidas de los distintos hilos de compilación se mezclan en la pantalla.
- `make modules` (utilizado en antiguos kernels, actualmente no es necesario): compila todos los módulos necesarios para satisfacer las opciones que hayan sido seleccionadas como módulo.
- `make modules install`: es utilizado en el proceso de compilación del kernel de Linux para instalar los módulos del kernel compilados en el sistema.
- `make install`: es utilizado en el proceso de compilación del kernel de Linux para instalar el kernel compilado en el sistema.

▼ Práctica 2

▼ Kernel

Tiene muchas líneas de código y las mayor parte son drivers.

El kernel puede ser monolítico o micro kernel.

- Monolítico: hay un único gran componente donde se linkean los demás y comparten espacio de direcciones. Tienen memoria compartida. Esto

permite más performance si hay mal hardware.

- Microkernel: Los componentes del kernel se sitúan en distintos procesos de usuario. La comunicacion entre procesos es parte del kernel por lo que hay muchos cambios de modo. El kernel es chiquito entonces es más fácil de entender, actualizar y optimizar. Tiene baja performance.

▼ System calls

Son llamadas al kernel para ejecutar una acción especifica ya sea para manejar un dispositivo o ejecutar una instrucción privilegiada. Su propósito es proveer una interfaz común para lograr portabilidad.

Su funcionalidad se ejecuta en modo Kernel pero en contexto del proceso.

▼ Invocación de syscall

Se puede hacer usando un wrapper de `glibc` o de manera explícita:

```
struct timeval tv;
/* usando el wrapper de glibc */
gettimeofday(&tv, NULL);
/* Invocacion explicita del system call */
syscall(SYS_gettimeofday, &tv, NULL);
```

▼ Interrupciones

La manera en que se ejecutan las syscalls depende de cada 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,

▼ Desarrollo de syscall

1. Se identifican con un número único: **syscall number**.
2. Se agrega una syscall a la syscall table. En el archivo `<kernel_code>/arch/x86/syscalls/syscall_32.tbl` se guarda la tabla de system calls para la arquitectura de x86 de 32 bits. Por otro lado, el archivo `<kernel_code>/arch/x86/syscalls/syscall_64.tbl` guarda la tabla de system calls para la arquitectura de x86 de 64 bits.
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 es un comando que reporta las syscalls que realiza cualquier programa. Utilizándolo junto al parámetro `-f` tiene en cuenta, además, a los procesos hijos.

▼ Ejemplo de syscall

Crearemos un nuevo fichero .c bajo el directorio `<source>/kernel`.

```
#include <linux/syscalls.h> /* For SYSCALL_DEFINEi() */
#include <linux/kernel.h>
SYSCALL_DEFINE0(mysyscall)
{
  printk(KERN_DEBUG "Hello world\n");
  return 0;
}
```

Luego modificamos el Makefile: `obj-y = ... mysyscall.o ...`.

Para probarlo se puede simplemente llamar a la system call.

```
#include <linux/errno.h>
#include <sys/syscall.h>
#include <linux/unistd.h>
#include <stdio.h>
#define __NR_MYSYSCALL -num-syscall
int main() {
  printf("Invocando system call...\n");
  return syscall(__NR_MYSYSCALL);
}
```

Luego creamos un parche: `diff -urpN linux-5.16-modificado > patch-5.16.16-so2022` y ejecutamos con el comando: `make mrproper`.

▼ SO APIS

Los SO proveen interfaces para que los procesos accedan a un conjunto de funciones. En UNIX lo hace `libc`.

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).

▼ Módulos

Los módulos son códigos que extienden la funcionalidad del kernel, son cargados y descargados bajo demanda. Sin módulos sería monolítico.

▼ Comandos

- `lsmod`: lista los módulos cargados. Es el equivalente a: `cat /proc/modules`.
- `rmmod`: descarga uno o más módulos.
- `modinfo`: muestra información sobre el módulo.
- `insmod`: trata de cargar el módulo especificado.
- `depmod`: permite calcular las dependencias de un módulo. Junto al parámetro `-a` escribe las dependencias en el archivo `/lib/modules/version/modules.emp`.
- `modprobe`: emplea la información generada por depmod e información de `/etc/modules.conf` para cargar el módulo especificado. El archivo `/lib/modules/<version>/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.

  Cuando se carga un módulo de kernel con el comando `modprobe`, el sistema busca en el archivo `modules.dep` para determinar qué otros módulos dependen del módulo que se está cargando. Entonces, `modprobe` carga automáticamente estas dependencias en el orden adecuado para asegurarse de que todas las dependencias se resuelvan correctamente.

▼ Crear módulo

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 indicársele otras funciones: `module_init()` y `module_exit()`.

```
#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);
```

▼ 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 parámetros: `sudo insmod hello.ko user_name=guada`.

Luego se construye el Makefile y se compila:

```
obj-m += hello.o
all:
make -C /lib/modules/$(shell uname -r)/build M=$(pwd) modules
clean:
make -C /lib/modules/$(shell uname -r)/build M=$(pwd) clean
make
```

▼ Dispositivos y drivers

Un dispositivo es cualquier dispositivo de hardware.

Cada operación sobre un dispositivo se hace con un código especial para ese dispositivo, este código se llama “driver” y se implementa como un módulo.

Los dispositivos se pueden clasificar en dos: dispositivos de acceso aleatorio y dispositivos seriales.

Los drivers se clasifican en:

- Drivers de bloques: conjunto de datos persistentes. Leemos y escribimos de a bloques de 1024 bytes.
- Drivers de carácter: se accede de a 1 byte por vez, y 1 byte sólo puede ser leído por única vez.
- Drivers de red: tarjetas de red, WIFI, etc.

Un dispositivo se identifica con el major y minor number. Estos dispositivos se encuentran en `kernel_code/linux/Documentation/devices.txt`.

Se considera que cada dispositivo de hardware es un archivo (`device file`), y accedemos a ellos mediante operaciones básicas: `read`, `write` y `ioctl`. Por convención están en el `/dev`. Se crean mediante el comando `mknod` → 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.

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

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.

▼ Práctica 3

Las aplicaciones necesitan almacenar y recuperar información. Hay requerimientos fundamentales para almacenar info: poder almacenar mucha información, la información debe mantenerse disponible y se tiene que poder acceder concurrentemente a la información.

La información se almacena en archivos.

File System es la parte del SO que se encarga del manejo de los archivos.

▼ Archivo

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

Un file system permite manejar archivos y su organización en directorios, desde creación, eliminación, modificacion hasta busqueda.

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.

▼ Ext2

Es un filesystem extendido. Utiliza block groups, todos son del mismo tamaño menos el último.

Los block groups reducen la fragmentación y aumentan la velocidad de acceso.

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).

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.

Tiene 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.

La contra es que hay muchas escrituras en el disco.

▼ Ext4

Ext4, ***Fourth Extended FileSystem***, es la evolución de Ext2.

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**, es decir, descriptor que representa un rango contiguo de bloques físicos. 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”.

▼ XFS

Filesystem de 64 bits. Dividido en regiones llamadas “allocation groups”. 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

ProcFS es un 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. En Linux 2.6 se implementó un nuevo sistema de archivos virtual llamado “Sysfs”.

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.

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.

▼ RAID

RAID permite usar multiples discos para que el sistema de discos sea mas rapido, 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.

▼ Nivel 0

Array de discos con “striping”, proceso de dividir datos en bloques y esparcirlos en muchos dispositivos de almacenamiento, a nivel de bloque. La capacidad del RAID es la sumatoria de la capacidad de los discos participantes.

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.

▼ Nivel 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.

▼ Nivel 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.

▼ Nivel 6

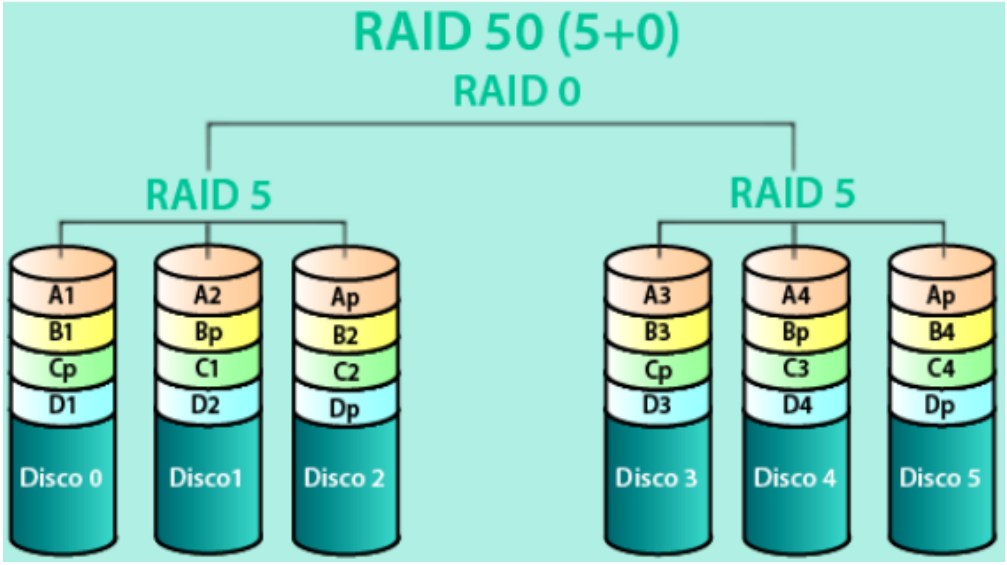
Striping a nivel de bloque y doble paridad distribuida. Recomendado cuando se tienen varios discos. Se requieren mínimamente 4 discos.

Alta tolerancia a fallos (hasta dos discos). Operaciones de escritura más lentas debido al cálculo de doble paridad.

▼ Niveles híbridos

Es posible combinar niveles:

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



- RAID 100

▼ DDP

Como los discos son cada vez más grandes, las técnicas de RAID son lentas.

Cuando hay una falla de disco en RAID se lee de los restantes discos, se recomputa paridad y el resultado se escribe en el nuevo disco (bottleneck). Consume mucho tiempo. Performance degradada.

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.

▼ LVM

Provee un método más flexible para alocar espacio en los dispositivos de almacenamiento masivo.

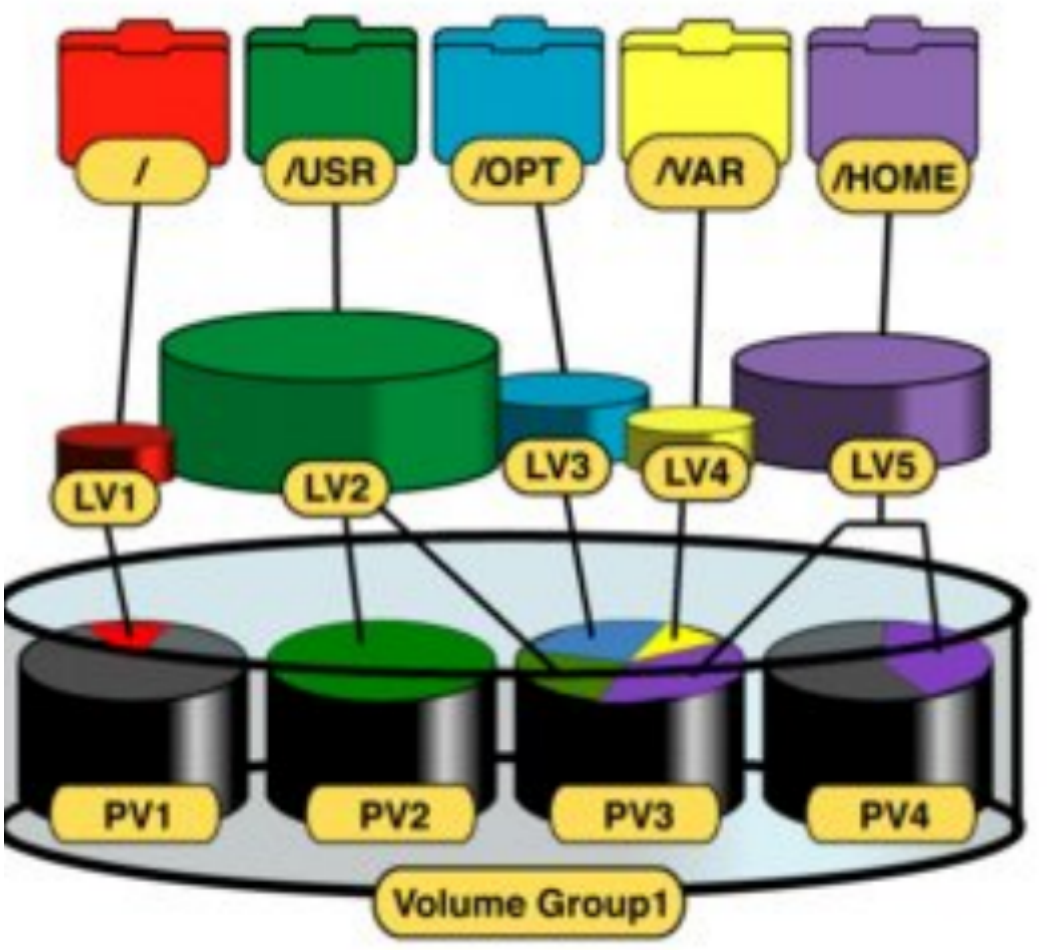
Provee una capa de abstracción entre el almacenamiento físico y el file system.

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.

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 basica en los LVs. Puede ser de distinto tamaño al del PE.



▼ Snapshot

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 genero 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.

“Copy-on-Write (CoW)”: datos son copiados al snapshot cuando se modifican.

▼ CoW

Usado en nuevos file systems para: Proveer consistencia en los datos y metadatos y Snapshots prácticamente instantáneos.

Modificar una archivo en file systems tradicionales reescriben datos en el mismo lugar.

CoW escribe los datos en una nueva ubicación, luego se actualizan los metadatos.

▼ BTRFS

Es un sistema de archivos basado en CoW.

El tamaño máximo de volumen es 16EiB. El tamaño máximo de archivo es 16EiB. La cantidad máxima de archivos es 2^64.

Utiliza extents. Permite agregar o remover dispositivo de bloques, agrandar o achicar volúmenes y desfragmentación online.

Tiene alocación dinámica de inodos.

Brinda soporte integrado de múltiples dispositivos, como son el RAID 0, RAID 1, RAID 5 y RAID 6.

Realiza checksum de datos y metadatos con CRC32. Permite comprimir archivos con zlib y LZO.

Otorga subvolúmenes y snapshots.

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.

Es posible la reduplicación, pero no se realiza online. Son herramientas adicionales.

▼ 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

▼ Comandos

- `mkfs.btrfs /dev/sdb`: crea un file system BTRFS en una partición
- `mkfs.btrfs /dev/sdb /dev/sdc`: crea un file system BTRFS sobre dos particiones
- `mkfs.btrfs -d raid1 -m raid1 /dev/sdb /dev/sdc`: crea un RAID1 espejando esas dos particiones
- `btrfs device add /dev/sdd /mnt/disco1`: agrega un nuevo dispositivo de bloques a un file system montado
- `btrfs filesystem show o btrfs filesystem df`: para ver las particiones de tipo BTRFS
- `btrfs-convert /dev/sdXX`: convierte un file system Ext3/4 a BTRFS

▼ Práctica 4

▼ Service isolation - chroot

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 pseudofilesystem llamado `cgroups`. Proporciona control de grano fino en alocación, priorización, denegación y monitoreo de recursos. Limita recursos, prioriza grupos, medicion de uso de recursos y permite controlar la vida de los procesos. Los procesos son ajenos a cgroups.

▼ cgroups v1

Asocia un conjunto de tareas con un conjunto de limites para uno o más subsistemas, 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 organizos en un árbol. Cada proceso puede pertenecer a un cgroups dentro de una jerarquía. Por cada una, la estructura de directorios refleja dependencia de cgroups.

Cada cgroup se representa por un directorio en una relacion padre-hijo. Los directorios tienen archivos que pueden ser escritos o leídos.

En cada nivel de la jerarquia se pueden definir atributos.

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 all/controladorespecial none/cgroup /directorio`.

Un directorio se puede desmontar si no tiene hijos y se hace con `umount y el directorio`.

▼ cgroups v2

Es el reemplazo de v1. Una jerarquía no puede estar en ambos cgroups simultáneamente.

Todos los controladores son montados en una jerarquía unificada. No se permiten procesos internos, a excepción del cgroup root. Los procesos deben asignarse a cgroups sin hijos.

No se puede especificar un controlador en especial para montar.

Cada jerarquia tiene el archivo controllers que indica los controllers disponibles para ese cgroup y subtree que es la lista de controladores habilitados en el cgroup.

Se habilitan controladores con + y se deshabilitan con -.

Inicialmente solo el cgroup root existe, todos los demás son hojas.

Cada cgroup tiene un archivo llamado procs.

▼ Namespace Isolation

Permite abstraer un recurso global del sistema para que los procesos dentro de ese namespace piensen que tienen una única instancia de ese recurso.

Limitan lo que un proceso usa y ve.

Toda modificación de ese recurso queda dentro del namespace.

Un processo puede estan en un namespace a la vez.

Cuando el proceso abandona o finaliza se elimina el namespace.

Un proceso puede usar alguno, todos o ningun namespace del padre.

Algunos son: IPC, network, mount, PID, Cgroup, etc.

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.

Mount permite aislar tabla de montajes.

▼ Linux containers

Los containers son 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.

Cada instancia es como una MV. Por fuera, son procesos normales del SO.

Los containers son eficientes, bootean rapido y no necesitan software de virtualización.

▼ 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. Por cada container Docker crea un conjunto de espacios de nombres (entre ellos pid, net, ipc y mnt).
- 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 imagenes tienen capas que se montan una sobre otra, solo la ultima es R/W. Cada capa es una instruccion 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 q sea persistente:

- Volúmenes: se almacenan en una parte del filesystem administrada por docker.
- Bind mounts: esta en cualquier parte del filesystem y puede ser modificada por procesos que no son de docker

▼ Comandos básicos

- docker pull imagen: descarga la imagen
- docker run imagen: ejecuta la imagen
- docker image build -t NOMBRE: crea una imagen a partir de un dockerfile
- docker push NOMBRE usuario/repositorio: sube la imagen al dockerhub 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

▼ Práctica 5

▼ Docker compose

Los sistemas estan formados por varias partes que necesitan comunicarse. Docker-compose es una herramienta que facilita hacer el despligue 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 archivo compose que tiene los recursos que se necesitan.

Docker compose es el binario que interpreta los compose file.

Un compose file esta escrito en YAML y tiene las caracteristicas de los contenedores a ejecutar, tiene toda la configuración. Se lo llama: docker-

compose.yaml. Tiene: version, services (toda la info del contenedor como el puerto, volumenes, enviroment, imagen, etc)

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 iniciaran los distintos contenedores con las características definidas para cada uno.

Docker compose es simple, replicable, compartible y versionable.

▼ Comandos básicos

- `docker-compose -v`: verificar versión instalada
- `docker-compose create`: crear todos los contenedores
- `docker-compose up`: iniciar todos los contenedores
- `docker-compose down`: frenar los contenedores
- `docker-compose up -d`: iniciar contenedores en background
- `docker-compose ps`: listar contenedores iniciados
- `docker-compose rm`: eliminar todos los contenedores
- `docker-compose logs`: ver los logs de los contenedores