01 - Lenguajes de programación:

Lenguaje de programación: Es un lenguaje especial, que especifica su propia sintaxis, reglas y conjuntos de palabras claves, utilizado para comunicar construcciones a una máquina.

En la segunda generación de computadoras (1953 – 1962) surgen los primeros lenguajes ensamblados, esto da lugar

Primera generación de computadoras (1938 – 1952) -> La lógica de un sistema se almacenaba de forma cableada. Surgen las tarjetas perforadas como medio de almacenamiento, y esto da origen a la **primera generación** de lenguajes de programación (Sin abstracción, lo que se escribía era en lenguaje maquina y se ejecutaba directamente en el hardware).

Segunda generación de computadoras (1953 – 1962) -> Surgen los primeros lenguajes ensamblados, esto da lugar a la **segunda generación** de lenguajes de programación.

Por último -> A finales de los años 50, el desarrollo de nuevos lenguajes de programación corresponde a querer abstraer al programador del hardware del sistema. Esto corresponde a la **tercera generación** de lenguajes de programación.

Niveles de abstracción en los lenguajes de programación:

Se clasifican con respecto al hardware. Por lo tanto, hay lenguajes de programación de bajo nivel y de alto nivel.

Bajo nivel -> **Lenguaje maquina** (es el lenguaje con el que trabaja el hardware, y de hecho es el único con el que puede trabajar, es una secuencia de dígitos binarios, NO posee abstracción alguna) y **lenguajes ensamblados** (consisten en una serie de instrucciones que se ejecutan directamente sobre el microprocesador y sus componentes) El hardware solo ejecuta instrucciones almacenadas en lenguaje máquina, debe existir un componente que se encarga de traducirlas, ese componente se llama assembler (<**ensamblador>**).

Código en lenguaje ensamblado ----> Assembler -----> Lenguaje máquina.

Alto nivel -> Tienen como principal característica la abstracción del hardware del equipo. No es necesario conocer la arquitectura del sistema cuando se trabaja con ellos. También se pueden identificar distintos niveles de abstracción. Por ejemplo, algunos lenguajes de programación de alto nivel nos prevén herramientas para manejar aspectos del hardware, como el lenguaje C permite el manejo de punteros (es un lenguaje de alto nivel, pero de abstracción muy baja). A diferencia de JavaScript, cuya sintaxis se encuentra abstraída del hardware.

Lenguajes compilados -> Código fuente de la aplicación -> Compilador -> Código fuente en lenguaje máquina.
Lenguajes interpretados -> Código fuente de la maquina -> Interprete -> Instrucciones en lenguaje de máquina.

El sistema sólo es capaz de ejecutar las instrucciones almacenadas en lenguajes de máquina, los lenguajes de programación de alto nivel necesitarán de un componente que “traduzca” las instrucciones escritas por el programador, al lenguaje de máquina. Este componente puede ser un **compilador** o un **intérprete**. Esto hace que dentro de los lenguajes de programación de alto nivel existan dos grandes tipos:

Los lenguajes de programación **compilados** son aquellos que se traducen enteramente al lenguaje de máquina para ser ejecutados directamente en el hardware. La **principal ventaja** de los lenguajes **compilados**, es la velocidad de ejecución. Dado que el compilador traduce el código fuente y lo almacena en código de máquina, la ejecución de estos archivos resulta mucho más rápida y eficiente que la ejecución de instrucciones en un lenguaje interpretado. Entre las **desventajas** que podemos mencionar de los lenguajes compilados, se encuentra la necesidad de compilar el código fuente antes de poder ejecutar la aplicación.

Algunos ejemplos de lenguajes de programación compilados son C, C++.

Los lenguajes de programación **interpretados** son aquellos en los que necesitan de un software intermedio llamado intérprete que será el encargado de ejecutar las instrucciones. Generalmente, el intérprete sí está almacenado en lenguaje de máquina. La ventaja de los lenguajes de programación **interpretados** es que no necesitan ser traducidos a código de máquina para ejecutarse. O al menos, no directamente.

La principal **desventaja** de este tipo de lenguajes es la velocidad de ejecución. El código nunca se encuentra almacenado en lenguaje de máquina, cada ejecución requiere de la "traducción" de las instrucciones. Al no requerir una compilación cada vez que se ejecutan las aplicaciones, el desarrollo de las mismas se realiza de una manera más ágil.

Algunos ejemplos de lenguajes de programación interpretados son PHP, Perl, Ruby, JavaScript, Python.

Clasificación de los lenguajes de programación dependiendo del paradigma

Dentro del ámbito del desarrollo de software, podemos encontrar distintos paradigmas o "formas" de programación. Existen paradigmas orientados a la metodología de trabajo, que imponen reglas o formas de trabajar desde lo práctico. Por otro lado, podemos identificar paradigmas de programación que dependen del lenguaje. Algunos ejemplos de paradigmas son la programación orientada a objetos, programación procedural, programación funcional, etc. Las características de cada lenguaje de programación serán las que ubiquen al mismo dentro de alguno de los paradigmas

Resumen

Los lenguajes de programación son la forma en la que el programador puede dar las instrucciones a la computadora para que la misma realice distintas operaciones. Los lenguajes de programación pueden clasificarse por su nivel de abstracción, encontrando en el nivel más bajo el lenguaje de máquina y los lenguajes de ensamblado. Sobre los lenguajes de programación de bajo nivel, en un nivel superior, encontramos los lenguajes de programación de alto nivel. Estos lenguajes son aquellos que buscan abstraer al programador del hardware, es decir que la persona no necesite desarrollar para distintos dispositivos la misma aplicación. Dentro de los lenguajes de programación de alto nivel podemos identificar dos grandes grupos: Lenguajes de programación compilados y lenguajes de programación interpretados. Por último, dentro de la programación existen distintos paradigmas, que corresponden a distintas "formas" de programar. Las características de cada lenguaje de programación son las que ubican a cada uno en alguno de los paradigmas.

-

02 - Elementos fundamentales de los Sistemas Operativos

¿Qué es el Kernel?

El **Kernel**, o núcleo del sistema operativo es el componente responsable de realizar el manejo de las funciones de bajo nivel de la computadora, por ejemplo:

- Ubicar un programa en la memoria
- Asignar el tiempo de CPU de un programa
- Hacer de interfaz entre el software y los dispositivos
- Permitir la interacción entre diferentes programas

Cada sistema operativo tiene su propio **Kernel**, es decir que, Mac OS X tiene un **Kernel** distinto al que podemos encontrar en cualquier versión de Windows o las distribuciones de Linux. Cada uno está diseñado de una manera distinta, por lo que provee las funcionalidades de una manera diferente, aunque la funcionalidad general sea la misma.

Además del kernel, ¿Qué diferencia un SO de otro?

- **Intérprete de línea de comandos o command-line shells:** Hace algunos años, los usuarios interactuaban con la computadora exclusivamente escribiendo comandos. Se utilizaban para renombrar archivos, ejecutar programas o realizar cualquiera de las funciones que permitía el sistema operativo. Aunque hoy en día la mayoría de los usuarios no utiliza (y en algunos casos ni conoce) la línea de comandos, esta interfaz sigue siendo imprescindible en muchos casos. Existen distintos tipos de **shells**, entre los cuales se encuentran **sh** (bourne shell), **bash** (bourne-again shell), **csh** (c shell), entre otros.

- **Interfaz gráfica:** La interfaz gráfica o GUI (Graphical User Interface) es una interfaz que proporciona un uso mucho más intuitivo de la computadora. En lugar de ejecutar comandos, una interfaz gráfica basa su manejo en íconos, menús, y punteros de mouse. Microsoft Windows y Mac OS tiene sus propias interfaces gráficas, mientras que Linux se basa en una interfaz gráfica conocida como X Windows System o simplemente X, utiliza paquetes de software conocidos como entornos de escritorio (GNOME, KDE, XFCE, etc).

- **Programas utilitarios:** Los sistemas operativos se instalan con varios programas utilitarios como calculadoras, calendarios, aplicaciones, etc. Estos programas varían entre los distintos sistemas operativos.

- **Librerías:** Las librerías son archivos que contienen las funciones básicas que utilizan las distintas aplicaciones. Existen librerías específicas para cada aplicación o para cada funcionalidad del sistema operativo. Las distribuciones basadas en GNU/Linux tienen todas las librerías ubicadas en un directorio llamado lib, pero, además, existe lib64 que almacena las mismas librerías, pero compiladas para la arquitectura de microprocesadores de 64 bits. Cuando un sistema contiene tanto los directorios lib y lib64 se dice que el sistema operativo es **Multilib**.

- **Programas varios:** Cada sistema operativo tiene su propio conjunto de aplicaciones de uso común como navegadores, software de ofimática, etc.

Uso de la interfaz gráfica

La mayoría de los usuarios prefiere utilizar interfaces gráficas, dada la simplicidad y la manera intuitiva en la que presenta la información. Es por eso que la mayoría de los sistemas operativos orientados al usuario común se instalan con una interfaz de usuario. GNU/Linux provee distintas interfaces gráficas, o, entornos de escritorio. Las opciones más comunes en lo que respecta a entornos de escritorio en Linux son: GNOME, KDE, XFCE, y Unity (Basado en GNOME y que actualmente es el escritorio por defecto de Ubuntu).

Comparando GNU/Linux con Unix

Unix es un sistema operativo creado en 1969 escrito en lenguaje ensamblado. En el año 1972, el sistema operativo completo se reescribió en lenguaje C y de ahí en más, fue posible la compilación para distintos dispositivos. El desarrollo de sistemas operativos basados en Unix se vio limitado debido a las demandas ocasionadas. En 1983, Richard Stallman decide crear GNU que comprenderá una copia de todas las aplicaciones que conformaban Unix, sin utilizar el código fuente del mismo. Para poder completar un sistema operativo completo, al proyecto GNU le faltaba un Kernel. Esto se solucionó en el año 1991 con el desarrollo del Kernel Linux por Linus Torvalds. El conjunto de GNU y Linux da como resultado **GNU/Linux**. GNU/Linux (muchas veces se dice Linux y esto en realidad está mal dicho, ya que **Linux** es el **Kernel** y **GNU** las **aplicaciones**) puede ser considerado con un miembro de la familia de sistemas operativos Unix, aunque técnicamente no lo sea.

Comparando GNU/Linux con Mac OS X

Licencia: Los sistemas operativos GNU/Linux son de código abierto, mientras que Windows se distribuye en base a una licencia comercial de software propietario.

Compatibilidad: Los fabricantes de hardware desarrollan los drivers específicamente para ser instalados en las diferentes versiones de Windows, aunque muchas veces también proveen de los drivers necesarios en su versión para GNU/Linux, ciertos componentes suelen funcionar mejor bajo Windows, ya que fueron fabricados específicamente para ese sistema operativo.

Aplicaciones: Al ser Windows el sistema operativo más instalado en equipos de escritorio y notebooks, la mayoría de las aplicaciones suele desarrollarse para esta plataforma. Sin embargo, existen variantes para ambos sistemas operativos.

Interfaz gráfica: Al igual que OS X, Windows posee su propia interfaz de usuario la cual es popular y amigable, de todas maneras, los entornos de escritorio disponibles para GNU/Linux fueron evolucionando y hoy pueden competir sin dudas con la interfaz de Windows.

Configuración y seguridad: Con configuración, nos referimos a la posibilidad de modificar el SO y adaptarlo a cada necesidad particular, en este punto, GNU/Linux es superior a Windows. Evidencia de esto es la innumerable cantidad de sistemas operativos basados en GNU/Linux y orientadas a distintos usos.

Para resumir, por casi dos décadas, Windows fue el sistema operativo dominante en lo que respecta a computadoras de escritorio y notebooks. Tanto en uso particular como en uso de oficina, los usuarios fueron familiarizándose con la interfaz de ese sistema operativo y muchas de las aplicaciones de uso general como Microsoft Office, etc. Si bien GNU/Linux podría ser utilizado en esos casos, el hecho de que los usuarios estén acostumbrados a utilizar Windows hace que el cambio hacia GNU/Linux sea una ardua tarea. Unix y GNU/Linux particularmente, han sido dominantes en el mercado de los servidores, debido a su manejo de la seguridad y los permisos, el rendimiento de los mismos, y las posibilidades de configuración y escalabilidad. Si bien Windows comercializa una versión dedicada a ser utilizada como servidor, nunca logró captar la mirada del mercado.

¿Qué es una distribución? Nos referimos como distribución al conjunto del Kernel de Linux, el conjunto de aplicaciones fundamentales que hacen al sistema operativo, y las configuraciones del mismo.

03 - Entendiendo los distintos tipos de licencia de software

¿Qué son las licencias de software?

El **software** es un tipo de propiedad intelectual que está gobernado por las **leyes de copyright** y en algunos países, se encuentran patentados. En general, el software de código abierto debe mucho a tres organizaciones particulares: the **Free Software Foundation** (FTF), the **Open Source Initiative** (OSI), y the **Creative Commons** (CC).

El **software Copyright** es, un reconocimiento legal de los derechos de copia que se aplican sobre cualquier cosa. La mayoría de las leyes de copyright fueron escritas mucho antes de que existieran las primeras computadoras, generalmente no se adecúan a las necesidades de las mismas

Las **licencias de software** pueden modificar los derechos de las **leyes de copyright** en mayor o menor medida.

*Las **licencias de software propietario** son aquellas que **restringen** aún más los derechos adquiridos por las leyes de copyright, mientras que las **licencias de código abierto** son aquellas que dan más **libertades** que las establecidas por las leyes de copyright.*

¿Qué es el software libre?

Es aquel que expone su código y no establece restricciones. No necesariamente es gratuito. Por ejemplo, el sistema operativo Red Hat. El código del mismo es libre, por lo que cada persona puede compilarlo cuantas veces quiera. De todas maneras, el sistema operativo ya compilado, es comercializado por la empresa Red Hat Inc. Que además provee soporte para el mismo.

Las libertades del software

- Libertad de **usar** el software para cualquier propósito
- Libertad de **examinar** el código fuente y modificarlo para adecuarse a las necesidades
- Libertad de **redistribuir** el software
- Libertad de **redistribuir** el software modificado

La filosofía que promueve la **Free Software Foundation** y las licencias inspiradas en ella, son comúnmente llamadas **copyleft**, ya que en vez de limitar la redistribución del contenido como definen las leyes de copyright, dan libertades de redistribuir y hacer prácticamente cualquier cosa con los contenidos.

The Free Software Foundation y la licencia GPL

Richard Stallman y la organización **Free Software Foundation** ya habían definido las libertades del software libre y qué es el software libre, pero no fue hasta que se creó la licencia GPL, que estas definiciones tuvieron un marco legal. **GPL** es la expresión legal de los principios que establece la Free Software Foundation.

GPL Versión 2 y GPL Versión 3: (la versión 1 cayó prácticamente en desuso). Estas dos versiones de la licencia GPL, **permiten aplicar las cuatro libertades del software libre a cualquier desarrollo que se realice**. Además, establecen explícitamente que cualquier desarrollo licenciado bajo GPL, también deben ser licenciados bajo GPL. Esto quiere decir que cualquier aplicación que surja de la modificación de un software bajo licencia GPL, también debe poseer la licencia GPL, sin excepción. Esta **licencia** fue utilizada por Linus Torvalds en el desarrollo de Linux, modifica las leyes y da el derecho a redistribuir tanto los archivos binarios como el código fuente de cualquier aplicación.

04 – Virtualización

¿Qué es la virtualización?

En informática, cuando se habla de **virtualización**, se hace referencia a la abstracción de algunos componentes físicos, en componentes lógicos. El primer antecedente de la virtualización de una computadora data de los años ’60 en IBM.

Una **máquina virtual** puede virtualizar todos los recursos de hardware, incluyendo el procesador, la memoria, el almacenamiento y hasta la conexión de red. Para poder realizar esto, debe existir un monitor de máquina virtual, comúnmente llamado **hypervisor**, que será el encargado de proveer el entorno necesario para la operación de la máquina virtual. Tiene las siguientes características:

- **Fidelidad:** El entorno creado por el hipervisor debe ser idéntico al de una máquina física.
- **Aislamiento:** Control completo de la máquina que está virtualizando, y debe aislar la misma del sistema que la está ejecutando.
- **Rendimiento**: No debe haber o debe haber poca diferencia de rendimiento entre una máquina virtual y su equivalente físico. (Es común que esta característica sea la que menos se cumpla).

Entendiendo la importancia de la virtualización

Básicamente se necesitaban muchas computadoras para el avance de nuevas tecnologías. Y por cada computadora, mucho hardware. Desafortunadamente, este hardware no es gratuito y de hecho muchas veces es costoso, además de que ocupa mucho espació y una infraestructura edilicia propicia, lo cual hace que los costos se eleven aún más. Pero es por todo esto que surge la **alternativa de la virtualización**. Sabiendo que la virtualización puede simular una computadora **lógica** (que hace lo mismo que una computadora **física**), dentro de una máquina **física**, podríamos tener una sola computadora **física**, que virtualice más de una computadora lógica**. Allí radica la principal ventaja de la virtualización.**

Definición y funciones del Hypervisor

El Hypervisor, anteriormente llamado monitor de máquina virtual, es una capa de software que reside entre la máquina virtual y el hardware del host. Sin el Hypervisor, las máquinas virtuales tratarían de comunicarse directamente con el hardware del host, haciendo que se produzca un caos entre las mismas y el sistema operativo del host. El Hypervisor se encargará de manejar las interacciones entre las máquinas virtuales y el hardware del host.

Hypervisor tipo 1: Es aquel que se ejecuta directamente sobre el hardware del host, sin un sistema operativo en el medio. Al no tener un sistema operativo como intermediario, puede tener una interacción directa con el hardware.

• **Ventaja:** Rendimiento mejor, se considera que este tipo de Hypervisor es más seguro. Al no existir un sistema operativo intermediario, no existirá la posibilidad de que este afecte de algún modo la ejecución del sistema virtualizado. Este tipo de Hypervisor requiere de menos procesamiento (porque no existe un sistema operativo intermedio que procese las interacciones). Esto permite que podamos ejecutar más máquinas virtuales al mismo tiempo.

Hypervisor tipo 2: Es una aplicación que se ejecuta sobre el sistema operativo. Por lo tanto, con este tipo de Hypervisor, el sistema operativo del host hará de intermediario, manejando las interacciones entre la máquina virtual y el hardware.

• **Ventaja:** Soporta un rango de hardware más amplio que el tipo 1, ya que hereda la compatibilidad del sistema operativo host. Es más fácil la instalación y configuración de este tipo de máquinas virtuales.

• **Desventaja:** El Hypervisor tipo 2 no es tan eficiente, debido a la capa intermedia que representa el sistema operativo del host. Por ejemplo, si el sistema operativo dentro de la máquina virtual necesita escribir información en el disco, no podrá acceder a él directamente. Deberá comunicarse con el sistema operativo del host y este será el encargado de comunicarse con el disco. Lo mismo sucede con todos los recursos como la placa de red, de sonido, etc.

05 - Línea de comandos y comandos principales

Entendiendo la línea de comandos

La línea de comandos, en GNU/Linux, Unix, OSX o Windows, es una terminal que permite al usuario ejecutar instrucciones en la computadora. La línea de comandos puede parecer complicada al principio, pero es una de las herramientas más potentes que tiene a mano cualquier desarrollador o administrador de sistemas.

Rutas o paths:

Una ruta, es la dirección de un archivo (o directorio) dentro del sistema de archivos. Las **rutas absolutas** son rutas que se escriben desde la raíz del sistema de archivos. Las **rutas relativas** son aquellas que no se indican desde el directorio raíz, sino que se escriben desde la "posición actual".

ls -> Ver el contenido de directorios.
-l Muestra los archivos en forma de columna. // **-a** Muestra todos los archivos (incluyendo los ocultos) // **-R** Listado recursivo de todos los archivos y subdirectorios. // **-i** Muestra el número de inodo // **-h** Muestra el espacio ocupado del archivo en MB, Kbyte, etc. // **-t** Ordena

Man -> Ver los manuales de los programas y los archivos de configuración.
Info -> Supera la información, nos permite navegar por medio de los enlaces como si fuera una página web.
Whatis -> Nos da una pequeña descripción de un comando y las secciones de man que podemos consultar.

Touch -> Si un archivo no existe este comando lo creará con el nombre especificado y con la fecha del momento. Por otro lado, si el archivo existe, le cambiará la fecha y hora.

Cp -> Nos permite copiar un archivo o directorio completo.

More / Less / Cat -> Muestra el contenido del archivo y solo podemos bajar de a una línea por vez, o avanzar por. Cat, cumple la misma función, pero el contenido completo se muestra en la pantalla.

rm -> nos permite borrar tanto archivos como directorios completos
mv -> nos permite tanto mover como renombrar archivos o directorios.
mkdir -> Permite crear un nuevo directorio
rmdir -> Eliminar directorios que estén vacíos. **-p** eliminará también la carpeta superior

… Existen más comandos (Buscar en el resumen especifico de comandos) …

06 - Estructura de directorios, archivos y puntos de montaje

Estructura de directorios

La **estructura de directorios** en GNU/Linux está ordenada siguiendo el estándar **FHS** (Filesystem Hierarchy Standard), creado y mantenido por la organización **Free Standards Group**. Este estándar define las bases para que tanto los programas del sistema, como los usuarios y administradores, sepan dónde encontrar lo que buscan

Podemos definir dos clasificaciones principales: **estáticos/dinámicos** o **compartidos/restringidos**.

• **Estáticos**: Contienen binarios, bibliotecas, documentación y otros archivos que no cambian sin intervención del administrador. Pueden estar en dispositivos de sólo lectura. No es necesario que se hagan copias de seguridad frecuentemente.
• **Dinámicos:** Contienen archivos que no son estáticos. Deben encontrarse en dispositivos que sean de lectura-escritura. Es necesario que se realicen copias de seguridad frecuentes.

• **Compartidos:** Contiene archivos que pueden utilizarse en otro sistema que no sea el mismo que los almacena.
• **Restringidos:** Contiene archivos que no pueden compartirse.

Ejemplo de archivos:
/bin/ -> Comandos/programas binarios esenciales (cp, mv, ls, rm, etc.),
/boot/ -> Archivos utilizados durante el arranque del sistema (núcleo y discos RAM)
/dev/ -> Dispositivos esenciales, discos duros, terminales, sonido, video, lectores dvd/cd, etc

Archivos en Linux

Tanto GNU/Linux / otro sistema operativo basado en Unix, es un sistema operativo completamente orientado a archivos. Todo se representa con un archivo, tanto los **datos**. como los **periféricos**. o incluso los **medios de comunicación**. Existen distintos tipos de archivos para cada caso: los archivos **directorios**, los archivos **comunes** y los **especiales**.

Archivos directorios: Son una instancia especial de los archivos normales. Los directorios listan las localizaciones de otros archivos, algunos de los cuales pueden ser otros directorios.

Archivos comunes: Son archivos que pueden contener cualquier tipo de dato: Texto, Audio, Imagen, Scripts, Librerías de programación, etc. Por defecto, nada permite diferenciar unos de otros. Linux no conoce la noción de extensión de archivo (aspecto que sí sirve para diferenciar un tipo de archivo de otro en Windows) como componente interno de la estructura del sistema de archivos, por lo que la extensión no tiene importancia y se considera simplemente parte del nombre.

Archivos especiales: Existen varios tipos de archivos especiales, pero principalmente sirven de interfaz para los diversos periféricos. Encontramos estos archivos especiales en el directorio **/dev**.

Nombre de archivos

Es necesario seguir reglas a la hora de nombrar archivos. Actualmente se puede llegar hasta **255** caracteres. Distingue entre **mayúsculas** y **minúsculas**, La mayoría de los caracteres (cifras, letras, ciertos signos, caracteres acentuados) son aceptados, incluyendo el **espacio**. A pesar de esto es necesario evitar ciertos caracteres reservados en la **shell** como **son: & ; () ~ / \ ` ?.**

Rutas de archivos

Las rutas permiten definir la **ubicación** de un archivo en el sistema de archivos. El nombre de la ruta es la concatenación, desde la raíz, de todos los directorios que se deben cruzar para acceder a él, que están separados cada uno por el carácter **/.** Ejemplo: **/home/usuario1/Documentos/Imagen.jpg** El ejemplo representa la ruta absoluta donde se ubica el archivo "Imagen.jpg".

- **Ruta absoluta:** Es aquella que se escribe desde el directorio raíz.
- **Rutas relativas:** Éstas se basan en la posición actual en el sistema de archivos.

Inodos y enlaces

Cada archivo en el sistema está representado por un inodo (Un bloque que almacena información de los archivos, de esta manera a cada inodo podemos asociarle un nombre) A simple vista pareciera que a un mismo archivo no podemos asociarle varios nombres, pero gracias a los enlaces esto es posible. Existen dos tipos de enlaces:

- **Enlaces físicos:** Es un nuevo nombre asociado a un archivo. Es una forma de identificar el mismo contenido con diferentes nombres. Este enlace no es una copia separada del archivo anterior sino un nombre diferente para exactamente el mismo contenido. El enlace aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de **archivo**.**txt**. Cualquier cambio que se haga se reflejará de la misma manera tanto para **archivo**.**txt** como para **nuevo_nombre**.**txt. El contenido del inodo no se eliminará mientras haya un enlace físico que le haga referencia**

- **Enlaces simbólicos:** Un enlace simbólico también puede definirse como una etiqueta o un nuevo nombre asociado a un archivo, pero a diferencia de los enlaces físicos, el enlace simbólico no contiene los datos del archivo, simplemente apunta al registro del sistema de archivos donde se encuentran los datos. Tiene mucha similitud a un acceso directo en Windows o un alias en OS X. Éste enlace también aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de archivo.txt, reflejando todos los cambios que se hagan tanto para archivo.txt como para nuevo_nombre.txt. **Cuando el archivo original es borrado o movido a una ubicación diferente el enlace dejará de funcionar y se dice que el enlace está roto.**

Después del arranque se pueden añadir más sistemas de archivos manualmente con el comando **mount**. Se usa para montar sistemas de archivos dentro de la estructura del árbol del sistema. El comando **mount** admite dos tipos de opciones, unos para el comando en sí, y otros para especificar opciones del sistema de ficheros.
mount [opciones] [dispositivo|directorio]

El comando **tar** es utilizado normalmente para empaquetar o desempaquetar archivos. La sintaxis del comando es:
tar [parámetros] [archivo1] [archivo2]

07 - Repositorios y paquetes

Un repositorio es una "**bodega**" donde se encuentran los **paquetes** a instalar en cualquier distribución. Los repositorios son servidores **ftp** o **http** (aunque también pueden ser locales) en donde se encuentran **todos los paquetes disponibles** para una distribución. Se habla de "**paquetes**" en vez de "programas" porque un paquete puede contener imágenes, librerías, código fuente, documentación, traducciones y desde luego programas. Cada distribución tiene sus propios repositorios y su forma de clasificarlos, y básicamente funcionan muy parecido.

Tipos de paquetes

• **Paquetes stable o estables:** Consta de software estable y bien probado, y solamente cambia cuando se incorporar correcciones importantes de seguridad.
• **Paquetes en prueba o testing:** Esta área contiene paquetes que se espera que sean parte de la próxima distribución estable. No goza de actualizaciones del equipo de seguridad en el mismo momento en que salen.
• **Inestables o unstable:** Contiene los paquetes más recientes. Una vez que un paquete ha cumplido las exigencias de estabilidad y calidad de empaquetado, será incluido en testing. Unstable tampoco está soportada por el equipo de seguridad.

Gestor de paquetes APT

APT (Adavance Packaging Tool) es el gestor de paquetes que se utiliza en distribuciones derivadas de Debian GNU/Linux: Crunchbang, Trisquel Ubuntu, Lubuntu, Linux Mint, etc… Es un sistema muy estable y muy fácil de usar a la vez.

# apt-get update	Actualizar la base de datos de paquetes.
#apt-get check	Validar el estado de los paquetes
# apt-get upgrade	Actualizar paquetes
# apt-get dist-upgrade	Actualizar la distribución
# apt-get install	Instalación de un paquete
# apt-get --reinstall	Reinstalar un paquete
# apt-get -f install	Resolver dependencias
# apt-get remove	Remover paquetes
# apt-get --purge remove	Remover los archivos de configuración también
# apt-get clean	Limpiar los paquetes bajados
# apt-get autoclean	Limpiar paquetes no usados
# apt-get autoremove	Eliminar librerías innecesarias
# apt-cache search paquete	Buscar paquetes
# apt-cache policy paquetes	Obtener sobre un paquete instalado la versión y el candidato
# apt-cache showpkg paquetes	Obtener información de un paquete instalado
# dpkg -i	Instalar un paquete que tengamos en nuestro pc (un archivo .deb)
# dpkg -l | more	Listar los paquetes instalados
# dpkg -L mc | more	Ver contenido de los paquetes
# dpkg -r	Borrar un paquete
# which ls --- # **dpkg -S /bin/ls**	Buscar a qué paquete pertenece un comando.
# dpkg --configure paquete	Configurar un paquete
# dpkg --configure -a	Reconfigurar todo

8 – Memorias

Las computadoras tienden a acceder al almacenamiento en formas particulares. De hecho, la mayoría del acceso a almacenamiento tiende a exhibir uno (o ambos) de los siguientes atributos

• **El acceso tiende a ser secuencial:** Si el CPU accede a la dirección N, es muy probable que la dirección N+1 sea la próxima a acceder, en orden — una instrucción tras la otra
• **El acceso tiende a ser localizado:** si se accede a la dirección X, es muy probable que otras direcciones alrededor de X también serán accedidas en el futuro. Estos atributos son cruciales, debido a que permite que unidades de almacenamiento pequeña y más rápida, coloque efectivamente en memoria temporal almacenamiento más grande y lento.

Distintos tipos de memorias

Las computadoras de hoy utilizan una variedad de tecnologías de almacenamiento. Cada tecnología está orientada hacia una función específica, con velocidades y capacidades en combinación. Estas tecnologías son:
- Registros de CPU
- Memoria caché
- RAM
- Discos duros
- Almacenamiento fuera de línea para respaldos (cintas, discos ópticos, etc.)

En términos de capacidades y costos, estas tecnologías forman un espectro. Por ejemplo, los registros de CPU son:
- Muy rápidos (tiempos de acceso de unos pocos nanosegundos)
- Baja capacidad (usualmente menos de 200 bytes)
- Capacidades de expansión muy limitadas (se requiere un cambio en la arquitectura del CPU)
- Costosas

Sin embargo, en el otro lado del espectro, **el almacenamiento fuera de línea es**:
- Muy lento (tiempos de acceso se miden en días, si la media de respaldo debe ser entregada sobre largas distancias)
- Capacidad muy alta (10s - 100s de gigabytes)
- Capacidades de expansión prácticamente ilimitadas (limitadas por el espacio físico, para hospedar la media de respaldo)
- Bajo costo

Usando diferentes tecnologías con diferentes capacidades, es posible afinar el diseño del sistema para un máximo rendimiento al costo más bajo posible.

Registros de la CPU

Se ejecutan a la misma velocidad que el resto del CPU; de lo contrario habría un cuello de botella grave sobre el rendimiento completo del sistema. La razón para esto es que casi todas las operaciones realizadas por el CPU envuelven registros de una forma u otra. El número de registros de CPU (y sus usos) dependen estrictamente en el diseño arquitectónico del CPU mismo. No hay forma de cambiar el número de registros de CPU, solamente puede migrar a un CPU con una arquitectura diferente. Por estas razones, el número de registros de CPU se puede considerar como una constante, ya que sólo pueden cambiarse con mucho dolor y grandes costos.

Memoria caché

El propósito de la memoria caché es actuar como una memoria temporal entre los registros de CPU, limitados y de gran velocidad y el sistema de memoria principal, mucho más grande y lento — usualmente conocido como RAM1. La memoria caché tiene una velocidad de operación similar a la del CPU mismo, por eso cuando el CPU accede a datos en la caché, no tiene que quedarse esperando por los datos.

Cuando se leen datos desde la RAM, el sistema de hardware verifica primero para determinar si los datos deseados están en caché. Si los datos están en caché, estos son recuperados rápidamente y utilizados por el CPU. Sin embargo, si los datos no están en caché, estos se leen desde la RAM y, mientras se transfieren al CPU, también se colocan en caché (en caso de que se necesiten más tarde).

La caché "write-through" o inmediata es un poco más simple de implementar; por esta razón es la más común. La caché "write-back" es un poco más complicada; además de almacenar los datos, es necesario mantener cierto tipo de mecanismo que sea capaz de notificar que los datos en caché están al día o "limpios" (los datos en caché son los mismos que los datos en RAM), o que están "sucios" (los datos en caché han sido modificados, lo que significa que los datos en RAM ya no están actualizados). También es necesario implementar una forma de vaciar periódicamente entradas "sucias" en caché de vuelta en RAM.

Niveles de caché

Los subsistemas de caché en los diseños de computadoras de hoy día pueden ser de niveles múltiples; esto es, puede haber más de un conjunto de caché entre el CPU y la memoria principal. Los niveles de caché a menudo están enumerados, con los números menores más cercanos a la CPU. Muchos sistemas tienen dos niveles de caché:

- La **caché L1** a menudo está ubicada en el chip del CPU mismo y se ejecuta a la misma velocidad que el CPU

- La **caché L2** usualmente es parte del módulo de CPU, se ejecuta a las mismas velocidades que el CPU (o casi) y normalmente es un poco más grande y lenta que la caché L1

- Algunos sistemas (normalmente servidores de alto rendimiento) también tienen **caché L3**, que usualmente forma parte del sistema de la tarjeta madre. Como puede imaginarse, la caché L3 es más grande (y casi con seguridad más lenta) que la caché L2

Memoria principal – RAM

La RAM resuelve la mayoría del almacenamiento electrónico en las computadoras de hoy en día. La RAM es utilizada tanto para **almacenar datos** como para **almacenar los programas en uso**. La velocidad de la RAM en la mayoría de los sistemas actuales está entre la velocidad de la memoria caché y la de los discos duros y está mucho más cercana a la velocidad de la primera que a la segunda. La operación básica de la RAM es en realidad bien sencilla.

En el nivel más bajo, están los chips de RAM — circuitos integrados que "recuerdan". Estos chips tienen cuatro tipos de conexiones con el mundo externo:
- Conexiones de energía (para operar la circuitería dentro del chip)
- Conexiones de datos (para permitir la transferencia de datos hacia adentro y fuera del chip)
- Conexiones de lectura/escritura (para controlar si los datos se almacenaran o se recuperaran desde el chip)
- Conexiones de direcciones (para determinar si los datos en el chip serán leídos/escritos)

He aquí los pasos requeridos para almacenar datos en RAM:
- Los datos a almacenar se presentan a las conexiones de datos.
- La dirección en la que los datos se almacenaran se presenta a las conexiones de dirección.
- La conexión de lectura/escritura se coloca en modo de escritura.

La recuperación de datos es también muy directa:
- La dirección de los datos deseados se presenta a las conexiones de direcciones.
- La conexión de lectura/escritura es colocada a modo de lectura.
- Los datos deseados son leídos desde las conexiones de datos

Discos duros

Los discos duros son no-volátiles (los datos se mantienen allí, aún después que se ha desconectado la energía) Debido a esto, los discos duros ocupan un lugar muy especial en el espectro del almacenamiento. Su naturaleza no-volátil los hace ideal para el almacenamiento de programas y datos para su uso a largo plazo.

No es posible ejecutar los **programas** directamente cuando son almacenados en discos duros. Ellos deben primero ser leídos a la RAM. Los discos duros son de al menos un orden de magnitud más lento que todas las tecnologías electrónicas utilizadas para caché y RAM. La diferencia en velocidad es debida principalmente a su naturaleza electromecánica.

Hay cuatro fases distintas que toman lugar durante cada transferencia de datos desde o hacia un disco duro.
- Movimiento del brazo de acceso (5.5 milisegundos)
- Rotación del disco (.1 milisegundos)
- Lectura/escritura de datos de cabezales (.00014 milisegundos)
- Transferencia de datos hacia/desde la electrónica del disco (.003 Milisegundos)

Fundamentos de la memoria virtual

Nunca hay suficiente RAM. Mientras que esta frase puede sonar al principio un poco cómica, muchos diseñadores de sistemas operativos han empleado una gran cantidad de tiempo tratando de reducir el impacto de esta limitación. Esto lo han logrado mediante la implementación de la memoria virtual — **una forma de combinar RAM con un almacenamiento más lento para darle al sistema la apariencia de tener más RAM de la que tiene instalada realmente.**

Memoria virtual en términos sencillos

Imaginemos que para ejecutar la aplicación, deben haber más de **15000** bytes de RAM disponible; un byte menos y la aplicación no será capaz de ejecutarse. Este requerimiento de **15000** bytes se conoce como el **espacio de direcciones de la aplicación**. En caso que no se cumpla con esa capacidad, el programa dará un error. **No hay memoria suficiente**. Con la **memoria virtual** el concepto del espacio de direcciones de las aplicaciones toma un significado diferente. En vez de concentrarse en cuanta memoria necesita una aplicación para ejecutarse, un sistema operativo con memoria virtual continuamente trata de encontrar una respuesta a la pregunta **"¿qué tan poca memoria necesita la aplicación para ejecutarse?".**

Si solamente una parte de la aplicación está en memoria en un momento dado, ¿dónde está el resto? El resto de la aplicación se **mantiene en disco**. Actúa como un **almacenamiento** de **respaldo** para la RAM; un medio más lento y también más grande que actúa como un "respaldo" para un almacenamiento más rápido y más pequeño.

Memoria virtual: Los detalles

Espacio de direcciones virtuales: Es el espacio de direcciones máximo disponible para una aplicación.
- Varía de acuerdo a la arquitectura del sistema y del sistema operativo, puesto que es la arquitectura la que define cuántos bits están disponibles para propósitos de direccionamiento
- También depende del sistema operativo puesto que la forma en que el sistema operativo fue implementado puede introducir límites adicionales sobre aquellos impuestos por la arquitectura.

La palabra "**virtual**" significa que este es el número total de ubicaciones de memoria direccionables disponibles para una aplicación, pero no la cantidad de memoria física instalada en el sistema, o dedicada a la aplicación en un momento dado.

Memoria secundaria. Conceptos fundamentales y administración

La memoria secundaria es aquella memoria no volátil y que sirve de soporte al sistema, almacenando todos los programas de manera permanente, y desde la cual se toman los programas que deben ser ejecutados, para luego moverlos a la memoria principal. Los discos rígidos son el dispositivo más utilizado como memoria secundaria. La alternativa principal es el almacenamiento de estado sólido. Si bien este no es un disco rígido puede ser manejado como un disco rígido regular.

¿Qué significa "particionar" un disco?

Particionar un disco significa **dividir un disco físico en varios discos lógicos**. Una partición es un conjunto de bloques contiguos dentro de un disco rígido físico, y cada partición es interpretada por el sistema operativo como discos independientes. El disco rígido, necesitará lo que se conoce como "**tabla de particiones**", para poder especificar qué sectores físicos del disco corresponden a cada partición.

¿Por qué tener múltiples particiones?

• Tener múltiples particiones permite "**encapsular**" los datos. Cada partición tendrá si propio sistema de archivos, que será independiente del resto de las particiones. En caso de corrupción, la pérdida de datos se limita a esa partición.

• Múltiples particiones pueden tener **formatos distintos**, variando, por ejemplo, el tipo de sistema de archivos o la cantidad de bloques en la que se divide lógicamente el disco. Mejora el rendimiento del sistema.

• Podemos **limitar el tamaño** que puede utilizar un **proceso** o un usuario. Si los datos del usuario completan el tamaño disponible, esto no hará que el sistema operativo se quede sin espacio para trabajar.

Existen distintos tipos de particiones:

• **Las particiones primarias:** Toman hasta cuatro de las ranuras de particiones en la tabla de particiones del disco duro.

• **Las particiones extendidas:** Fueron desarrolladas en respuesta a la necesidad de más de cuatro particiones por unidad de disco. Una partición extendida puede contener dentro de sí múltiples particiones, extendiendo el número de particiones posibles en una sola unidad de disco. La introducción de las particiones extendidas se generó por el desarrollo constante de las capacidades de los discos duros.

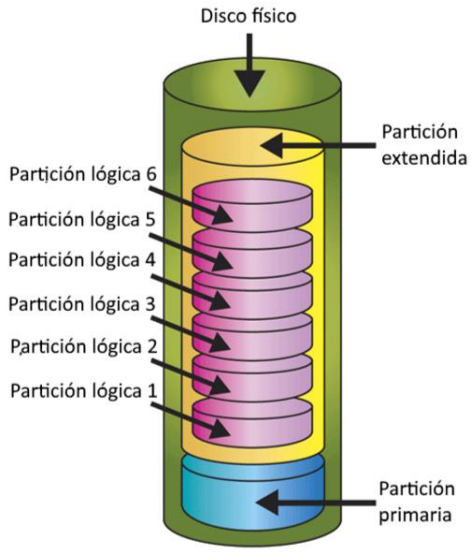

Diagrama representativo de los distintos tipos de particiones

Sistema de archivos

Un sistema de archivos es un **método** para **representar datos** en un dispositivo de almacenamiento masivo. Los sistemas de archivos usualmente incluyen las características siguientes:
•Almacenamiento de datos basado en archivos
• Estructura de directorio jerárquico (algunas veces llamado "carpeta")
• Seguimiento de la creación de archivos, tiempos de acceso y de modificación
• Algún nivel de control sobre el tipo de acceso permitido para un archivo específico
• Un concepto de propiedad de archivos
• Contabilidad del espacio utilizado

Almacenamiento de datos basado en archivos

Los sistemas operativos más antiguos con sistemas de archivos más primitivos permitían solamente caracteres alfanuméricos (y solamente mayúsculas) y únicamente nombres de archivos 8.3 (lo que significa un nombre de archivo de ocho caracteres, seguido de una extensión de tres caracteres).

Control de acceso

Es un área en la que los sistemas de archivos difieren dramáticamente. Algunos sistemas de archivos no tienen un modelo claro para el control de acceso, mientras que otros son mucho más sofisticados. En términos generales, la mayoría de los sistemas de archivos modernos combinan dos componentes en una metodología cohesiva de control de acceso:

•**Identificación del usuario:** El sistema de archivos (y el sistema operativo subyacente) primeramente debe ser capaz de identificar unívocamente a usuarios individuales. Algunos sistemas de archivos soportan es la creación de identificadores genéricos que se pueden asignar a uno o más usuarios

•**Lista de acciones permitidas:** Leer el archivo – Escribir el archivo – Ejecutar el archivo (A veces se pueden hacer mas acciones)

Tecnologías avanzadas de almacenamiento

Existen otras opciones más avanzadas además de los discos rígidos.

Almacenamiento accesible a través de la red

Combinando redes con las tecnologías de almacenamiento masivo puede resultar en una flexibilidad excelente para los administradores de sistemas.

Con este tipo de configuración se tienen dos beneficios posibles:

• **Consolidación del almacenamiento:** Se puede consolidar implementando servidores de alto rendimiento con conexiones de red de alta velocidad y configurados con grandes cantidades de almacenamiento rápido. Con la configuración apropiada, es posible suministrar acceso al almacenamiento a velocidades comparables al almacenamiento conectado directamente. Además se reducen los costos.

• **Administración simplificada:** Los servidores de almacenamiento centralizado también pueden hacer muchas tareas administrativas más fáciles. Por ejemplo, monitorizar el espacio libre es mucho más fácil cuando el almacenamiento a supervisar existe en un sólo servidor centralizado. Los respaldos también se pueden simplificar. Es posible hacer respaldos basados en la red para múltiples clientes, pero se requiere más trabajo para configurar y mantener.

Almacenamiento basado en RAID

Un disco duro con cuatro particiones se muere completamente: **¿qué pasa con los datos en esas particiones?** Las aplicaciones se vuelven más lentas debido a que el disco duro no puede procesar lecturas y escrituras más rápido. **¿Qué pasa entonces?** Existe una tecnología que puede resolver cada uno de estos problemas. El nombre de esta tecnología es **RAID** (Redundant Array of Independent Disks) **Formación de Discos Independientes Redundantes**. Es una forma para que discos múltiples actúen como si se tratasen de una sola unidad.

Niveles de RAID A

Al principio, fueron definidos cinco niveles RAID y los nombraron del "1" al "5." Luego, otros investigadores y miembros de la industria del almacenamiento definieron niveles RAID adicionales. No todos los niveles RAID eran igualmente útiles; algunos eran de interés solamente para propósitos de investigación y otros no se podían implementar de una forma económica.

Al final, había tres niveles de RAID que terminaron siendo ampliamente utilizados: **Nivel 0** /// **Nivel 1 (RAID I)** /// **Nivel 5 (RAID V)**

RAID 0

Se utiliza para doblar el rendimiento y para fusionar todos los discos duros en un sólo disco para aumentar la capacidad de almacenamiento. Es necesario tener 2 discos duros como mínimo. Por ejemplo, si tenemos dos discos que funciona a una velocidad alrededor de 20 Mb/s, al poner dos discos se duplicaría la velocidad es decir 40 Mb/s (2x20 Mb/s). Es una partición lógica cuyo tamaño es igual a la suma de los discos integrados en el sistema RAID.

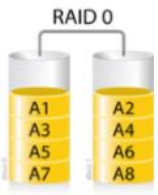

Figura 3

RAID 1

Es utilizado para garantizar la integridad de los datos, en caso de un fallo de uno de los discos duros, es posible continuar las operaciones en el otro disco duro sin ningún problema. No se mejora el rendimiento y no se suman el espacio de los discos como en RAID 0. El tipo de RAID 1 se llama comúnmente “mirroring” debido a que éste hace una simple copia del primer disco.

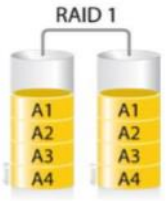

Figura 4

RAID 5

Trata de combinar los beneficios de RAID 0 y RAID 1, a la vez que trata de minimizar sus desventajas.

*Igual que RAID 0 consiste de múltiples unidades de disco, cada una dividida en porciones. Esto permite a una formación RAID 5 ser más grande que una unidad individual.

*Como en RAID 1, una formación RAID 5 utiliza algo de espacio en disco para alguna forma de redundancia, mejorando así la confiabilidad.

Una formación **RAID 5** debe consistir de al menos **tres discos idénticos** en tamaño (aunque se pueden utilizar más discos). Cada unidad está dividida en porciones y los datos se escriben a las porciones siguiendo un orden. Sin embargo, **no cada porción está dedicada al almacenamiento** de datos como en RAID 0. En cambio, en una formación con n unidades en ella, la enésima porción está dedicada a la paridad. Las porciones que contienen paridad hacen posible recuperar los datos si falla una de las unidades en la formación. La paridad en la porción x se calcula matemáticamente combinando los datos desde cada porción x almacenado en todas las otras unidades en la formación.

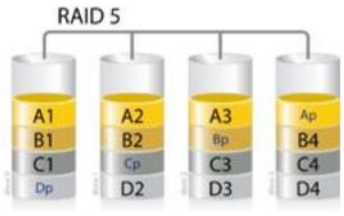

Figura 5

Niveles RAID anidados

Cada nivel tiene sus fortalezas y debilidades específicas. ¿Se podrían combinar de alguna forma, produciendo formaciones con todas las fortalezas y ninguna de las debilidades de los niveles originales?

• El orden en el que los niveles RAID son anidados pueden tener un gran impacto en la confiabilidad. En otras palabras, **RAID 1+0** y **RAID 0+1** no son lo mismo.

• **Los costos pueden ser altos** - La formación RAID 5+1 más pequeña posible, consiste de seis discos (y se requieren hasta más discos para formaciones más grandes). Esto es muy costoso.

Implementaciones RAID

Es obvio a partir de las secciones anteriores que RAID requiere "inteligencia" adicional sobre y por encima del procesamiento usual de discos de E/S para unidades individuales. Como mínimo, se deben llevar a cabo las tareas siguientes:
•**Dividir las peticiones** de E/S entrantes a los discos individuales de la formación
• Para RAID 5, **calcular la paridad** y escribirla al disco apropiado en la formación
• **Supervisar** los discos individuales en la formación y tomar las acciones apropiadas si alguno falla
• **Controlar** la reconstrucción de un disco individual en la formación, cuando ese disco haya sido reemplazado o reparado
• **Proporcionar** los medios para permitir a los administradores que mantengan la formación.

Hardware RAID

Una implementación de hardware RAID usualmente toma la forma de una tarjeta controladora de disco. La tarjeta ejecuta todas las funciones relacionadas a RAID y controla directamente las unidades individuales en las formaciones conectadas a ella. Con el controlador adecuado, las formaciones manejadas por una tarjeta de hardware RAID aparecen ante el sistema operativo anfitrión como si se tratasen de unidades de disco normales. La interfaz administrativa se implementa usualmente en una de tres formas:
• **Programas de utilerías** especializados que funcionan como aplicaciones bajo el sistema operativo anfitrión.
• Una **interfaz en la tarjeta** usando un puerto serial que es accedido usando un emulador de terminal.
• Una interfaz tipo **BIOS** que solamente es accesible durante la prueba de encendido del sistema.

Software RAID

Software RAID es RAID implementado como kernel - o software a nivel de controladores para un sistema operativo particular. Como tal, **proporciona más flexibilidad en términos de soporte de hardware** - siempre y cuando el sistema operativo soporte ese hardware, se pueden configurar e implementar las formaciones RAID.

Esto puede reducir dramáticamente el costo de implementar RAID **al eliminar la necesidad de adquirir hardware costoso especializado**. A menudo el exceso de poder de CPU disponible para los cálculos de paridad RAID exceden en gran medida el poder de procesamiento presente en una tarjeta controladora RAID. Por lo tanto, **algunas implementaciones de software RAID** en realidad **tienen mejores capacidades de rendimiento que las implementaciones de hardware RAID**.

Administración de volúmenes lógicos

Otra tecnología de almacenamiento avanzada es administración de volúmenes lógicos o logical volume management (LVM). Hace posible tratar a los dispositivos físicos de almacenamiento masivo como bloques de construcción a bajo nivel en los que se construyen diferentes configuraciones de almacenamiento. Las capacidades exactas varían de acuerdo a la implementación específica, pero pueden incluir la **agrupación del almacenamiento** físico, **redimensionamiento de volúmenes lógicos** y la migración de datos.

Agrupamiento de almacenamiento físico

Es la base para todas las implementaciones de LVM. Como su nombre lo implica, los dispositivos físicos de almacenamiento masivo se pueden agrupar de forma tal para crear uno o más dispositivos lógicos de almacenamiento. Los dispositivos lógicos de almacenamiento masivo pueden ser más grandes en capacidad que cualquiera de los dispositivos físicos de almacenamiento subyacentes.

Redimensionamiento de volúmenes lógicos

En una configuración de sistemas no LVM, el quedarse sin espacio significa - en el mejor de los casos - mover archivos desde un dispositivo lleno a uno con espacio disponible. A menudo esto significa la reconfiguración de sus dispositivos de almacenamiento masivo. Sin embargo, LVM hace posible incrementar fácilmente el tamaño de un volumen lógico. Asuma por un momento que nuestro parque de almacenamiento de 200GB fue utilizado para crear un volumen lógico de 150GB, dejando el resto de 50GB en reserva. Si el volumen lógico de 150GB se llena, LVM permite incrementar su tamaño (digamos por 10GB) sin ninguna reconfiguración física.

09 – Procesos

Proceso de arranque en GNU/Linux

Para poder empezar a hablar de procesos en GNU/Linux, y del manejo de los mismos, es necesario conocer cómo es el proceso de arranque del sistema. El proceso de arranque consta de cuatro etapas: 1. BIOS / 2. Bootloader / 3. Kernel 4. Init

Primera etapa: BIOS

Esta etapa inicia en el momento en que se enciende la PC. Es allí cuando el **BIOS (Basic Input Output System)** toma el control del sistema para poder **realizar operaciones básicas de hardware** (reconocimiento, prueba de la memoria, etc.). Una vez que el BIOS completa las operaciones, se encargará de cargar en memoria el **Bootloader** que inicia la segunda etapa. Es importante mencionar que esta **etapa es independiente del sistema operativo instalado**.

Segunda etapa: Bootloader

El bootloader o cargador de arranque, es un programa encargado de iniciar el sistema operativo instalado. Este programa se guarda en una porción del disco llamada MBR (Master Boot Record). El MBR ocupa solo 512 bytes en el disco, de los cuales 2 bytes corresponden al "magic number", 64 bytes a la tabla de particiones y 446 bytes al bootloader.

El bootloader será el encargado de iniciar el resto del sistema. Puede contener la información de distintos sistemas operativos. En GNU/Linux, existen dos bootloaders principales: **LILO y GRUB.**

• **LILO** sólo soporta hasta 16 sistemas operativos instalados y no tiene la posibilidad de iniciar alguno desde la red. No contiene una consola interactiva que permita modificar los parámetros de inicio de un determinado sistema operativo

• **GRUB** Tiene la posibilidad de manejar una cantidad ilimitada de sistemas operativos y de bootear por red, contiene una interfaz de línea de comandos interactiva que permite establecer parámetros al iniciar un sistema operativo.

Tercera etapa: Kernel

El kernel es el componente fundamental de cualquier sistema operativo, ya que es **el encargado de comunicar los programas que solicitan recursos y el hardware**. Asignará los tiempos de ejecución para cada programa, gestionará cada una de las tareas que realiza el sistema operativo, el hardware del equipo y el sistema de archivos utilizado.

El kernel puede presentarse desarrollado en distintas arquitecturas, **monolítico**, **modular** o **híbrido**.

•**Monolítico:** Implica que todos los módulos necesarios para controlar el hardware del equipo se encuentran cargados en un único archivo comprimido.
• **Modular**: Se construyen con cada módulo compilado como un objeto que puede ser cargado o descargado según sea necesario.

El proceso de carga del kernel se realiza en dos etapas: **etapa de carga y etapa de ejecución**. El kernel se **encuentra almacenado de forma comprimida en la memoria secundaria del sistema** (generalmente un disco rígido, pero puede ser un pendrive, etc.), por lo que la **etapa de carga del kernel se encargará de descomprimir el kernel y copiarlo** entero en la memoria principal (RAM). Además de la carga del kernel en la memoria principal, también se cargarán los drivers necesarios mediante un proceso llamado **initrd**. Este proceso creará un sistema de archivos temporal que sólo es utilizado durante la fase de carga.

Cuarta etapa: init

Un proceso es un programa que se ejecuta en un determinado momento en el sistema. Siempre hay una serie de procesos que se ejecutan sin la intervención del usuario y que hacen que el sistema sea utilizable.

Cada proceso contiene un conjunto de estructuras de datos y una dirección en la memoria principal. Existen dos tipos de procesos:
• **Procesos de usuario**: Son aquellos procesos ejecutados directamente por el usuario.
• **Procesos denomino:** Los procesos demonio son aquellos que no requieren la intervención del usuario y se ejecutan en un segundo plano.

Una vez que la etapa de carga y ejecución del **kernel** se completa, se iniciará el **proceso init**. Este proceso es ejecutado por todos los sistemas basados en Unix y **es el responsable de la inicialización de todos los nuevos procesos excepto el proceso swapper**.

Init se conoce como un **proceso planificador**, encargado de decir **qué proceso se ejecutará y cuáles serán copiados/borrados de la memoria principal**. En GNU/Linux, los procesos se ejecutan de forma jerárquica: cada proceso es lanzado desde un proceso "padre", por lo que el proceso lanzado se llama "hijo".

Atributos de un proceso

La **prioridad** de un proceso afecta al tiempo y al orden de ejecución de éste por parte del procesador. Un proceso de mayor prioridad que otro significa que en la ejecución de los dos procesos el sistema asignará un período de ejecución mayor para el de mayor prioridad y, además, lo elegirá más veces para ejecutarlo. Si no se indica nada al crear un proceso, éste toma la misma prioridad que la del proceso <padre>.

Control de terminal:

Son enlaces que determinan de dónde toman la entrada y la salida de datos y el canal de error los procesos durante su ejecución. Si no se usan redirecciones, se utilizan por defecto la entrada y salida estándar.

Creación y ejecución de un proceso

Cuando un proceso quiere crear un nuevo proceso, el primer paso consiste en realizar una copia de sí mismo mediante la llamada del sistema fork. La operación **fork** crea una copia idéntica del proceso padre, salvo en los siguientes casos:

• El nuevo proceso tiene un PID distinto y único.
• El PPID del nuevo proceso es el PID del proceso padre.
• Se asume que el nuevo proceso no ha usado recursos.
• El nuevo proceso tiene su propia copia de los ficheros descriptores del proceso padre.

•Los procesos "**dormidos**" esperan a que ocurra un determinado evento. Por ejemplo, entrada de datos, una conexión de la red, etc. No consumen tiempo del procesador.

• Los procesos "**intercambiados**" no se encuentran en la memoria principal del sistema, se vuelcan a la memoria de intercambio. Esto sucede cuando la memoria principal no dispone de suficiente espacio libre y los datos del proceso pasan continuamente de la memoria de intercambio a la principal como consecuencia, la ejecución del proceso se realiza de forma poco efectiva.

• Cuando un proceso está "**zombi**" no se puede volver a ejecutar, el espacio de memoria que utilizaba se ha liberado y sólo mantiene algunas de las estructuras de datos que les dan entidad a los procesos.

• Un proceso está "**parado**" cuando no se puede ejecutar. La diferencia entre un proceso "dormido" y otro "parado" es que este último no puede continuar ejecutándose hasta que reciba una señal CONT de otro proceso. Las siguientes situaciones llevan un proceso al estado de parada.

Señales

Las **señales** se utilizan para que un proceso suspenda la tarea que está realizando y se ocupe de otra. Cuando se envía una señal a un proceso, éste puede actuar de dos maneras: si el proceso dispone de una rutina específica para esa señal, llamada "**manejador**" o **handler**, la utiliza, en caso contrario, el **Kernel** utiliza el manejador por defecto para esa señal.

Utilizar un manejador específico para una señal en un proceso se denomina **capturar la señal**. Hay dos señales que no pueden ser ni capturadas ni ignoradas, **KILL** y **STOP**. La señal de **KILL** hace que el proceso que la recibe sea destruido. Un proceso que recibe la señal de **STOP** suspende su ejecución hasta que reciba una señal CONT.

10 – Usuarios, grupos y permisos

Linux fue diseñado para permitir múltiples usuarios al mismo tiempo. Para que estos usuarios no afecten las sesiones del resto de los usuarios, se tuvo que desarrollar un complejo modelo de permisos

Permisos de **lectura**, **escritura** y **ejecución**. Los permisos son los "derechos" que tiene un usuario o grupo sobre un archivo o directorio. Los permisos básicos son lectura, escritura y ejecución.

• Permiso de lectura (**read**): Este permiso hace que el contenido de un archivo sea visible, o, en el caso de los directorios, el que contenido del mismo sea visible.

• Permiso de escritura (**write**): Este permiso hace que el contenido de un archivo pueda ser modificado, mientras que para directorios permite realizar acciones sobre los archivos que contiene (borrarlos, agregar nuevos archivos, etc.)

• Permiso de ejecución (**execute**): En archivos, este permiso hará posible la ejecución del mismo. Para esto, el archivo deberá ser un script o un programa.

Listar los permisos de un archivo o directorio

Para listar los permisos de un archivo o directorio, basta con ejecutar el comando ls, junto con la opción -l. El resultado de la ejecución de dicho comando mostrará el contenido de un directorio de la siguiente manera:

-rw-r--r-- 1 root root 1031 Nov 18 09:22 /etc/passwd

Los primeros diez caracteres del ejemplo representan los permisos.

El primer carácter (que en este caso es un -) representa el tipo de archivo que está listando.
Los tipos de archivo son:
• **d** representa directorios
• **s** representa un archivo especial
• **-** representa un archivo regular

Los siguientes nueve caracteres representan puntualmente qué permisos tienen el propietario, los permisos que tienen aquellos usuarios que pertenezcan al mismo grupo que el propietario, y los permisos que tienen aquellos usuarios que no pertenecen al mismo grupo que el propietario. **El siguiente cuadro muestra cómo se descomponen los diez caracteres**, utilizando el ejemplo anterior:

-	rw-	r--	r--
Tipo de archivo	Permisos del dueño	Permisos del grupo del dueño	Permisos del resto de los usuarios

Podemos observar que los permisos del dueño son **rw-**. Esto quiere decir que el usuario tiene permisos de lectura (**r**), y permisos de escritura (**w**). Los permisos de ejecución, si los tuviera, estarían representados con una x luego del permiso de escritura. En este caso, como no posee permisos de ejecución, solo vemos un **-**. Siguiendo con la misma lógica, podemos decir que tanto los usuarios del grupo del propietario, como el resto de los usuarios, solo poseen permisos de lectura.

Administrando cuentas de usuario

El comando para crear un nuevo usuario estándar es **useradd**. La sintaxis es la siguiente: **useradd <nombre>** Las opciones que podemos utilizar son las siguientes:

-d <home_dir> Nos permite especificar cuál será el directorio principal del usuario que estamos creando.
-e <fecha> Permite especificar una fecha en la que expirará la cuenta del nuevo usuario.
-s Especifica el intérprete de línea de comandos que usará por defecto el nuevo usuario.
Por ejemplo: **useradd nuevousuario -s /bin/bash**

Otra manera de crear usuarios es utilizar el comando **adduser**. El mismo no se encuentra siempre instalado, por lo que hay que instalarlo como cualquier otro paquete. Este comando permitirá crear usuarios de una **manera más simple** ya que automáticamente creará el directorio principal, asignará un intérprete de línea de comandos y configurará el password.

Una vez creado el nuevo usuario, es necesario configurar un **password** para el mismo. Esto se hace utilizando el comando **passwd**. La sintaxis es la siguiente: **passwd <usuario>**

Para remover una cuenta de usuario, el comando a utilizar es **userdel**.
La sintaxis del mismo es: **userdel <usuario> /// userdel -r <usuario> (Esta opción elimina archivos y el directorio principal).**

Entendiendo sudo

El usuario **root**, es el usuario que **tiene los permisos para hacer cualquier cosa en el sistema**. Es por eso que, por motivos de seguridad, se suele utilizar **sudo**. Este comando permite a usuarios y grupos, tener acceso a comandos que no podría acceder normalmente. **Esto quiere decir que aquellos usuarios sudo podrán ejecutar comandos como si fuesen root sin realmente estar logueados como usuarios root**. En muchos casos será necesario instalar este paquete. Para que un usuario determinado sea pueda utilizar sudo, será necesario que el mismo esté presente en el archivo **sudoers**.

Trabajando con grupos

Los grupos en Linux

Son una manera de organizar los usuarios, principalmente como una medida de seguridad. El control de los grupos se administra a través del archivo **/etc/group**, que contiene la lista de grupos y sus miembros. Para cambiar el grupo con el que un usuario está asociado en un momento determinado, se utiliza el comando **chgrp**.

Cambiando permisos de archivos y directorios

Suponiendo que ejecutamos el comando **ls -s**, si la salida es la siguiente:

-rw-r--r-- 1 root root 1031 Nov 18 09:22 /etc/passwd

Comando chmod

El comando **chmod** nos permitirá cambiar permisos en archivos o directorios. Existen dos maneras de especificar nuevos permisos al utilizar el comando **chmod**, con letras o números (en base octal). Para utilizarlo con letras, hay que tener en cuenta lo siguiente:
El signo **+** agregará permisos y el signo **–** quitará permisos.

- **r** representa el permiso de lectura.
- **w** representa el permiso de escritura.
- **x** representa el permiso de ejecución.
- **X** representa el permiso de ejecución (sólo aplica a directorios).

EJ: **chmod +r <archivo>**

Permisos adicionales en archivos

Además de los permisos comunes, **lectura**, **escritura** y **ejecución**, hay algunos permisos adicionales que también pueden resultar útiles, que describen el comportamiento de los archivos y ejecutables en situaciones "**multiusuario**". Los adicionales son:

sticky bit (t): significa que solo el propietario (o root) pueden eliminar el archivo, independientemente de los permisos que tengan el resto de los usuarios. Para configurar el permiso **sticky bit** sobre un archivo ejecutamos los siguiente: **chmod +t <archivo>**

setuid bit (s): Este comportamiento es similar a la opción "ejecutar como", que permite ejecutar un programa como si fuera un usuario distinto. Suponiendo que el resto de los usuarios pertenecientes al **grupo1** tienen permisos de ejecución sobre el archivo **archivo1**.**sh**, éstos podrán ejecutar el archivo como si fueran el **usuario1.**

Cambiando el propietario de un archivo

Por defecto, el propietario de un archivo es aquel que **crea el mismo**, y por defecto, el grupo asociado es **grupo en que esté registrado el usuario al momento de crear el archivo**. Para cambiar el propietario de un archivo, se utiliza el comando **chown**.

-rw-r--r-- 1 administrador1 grupoadmin 1031 Nov 18 09:22 /home/admin/lista.txt

Si se quisiera cambiar el propietario de tal archivo (que actualmente es administrador1, y está bajo el grupo grupoadmin) por administrador2, el comando a ejecutar sería el siguiente:

chown administrador2:grupoadmin /home/admin/lista.txt

11 – Protocolos

Los **protocolos** son **mecanismos** que definen las **reglas** y **convenciones** para la **comunicación entre dispositivos**. Los protocolos incluyen tanto los mecanismos para **identificar** y **realizar** conexiones dentro de una red, así como también especifican las reglas y el formato con el que deben transmitirse los datos. Existen protocolos que, además, incluyen soporte para la compresión de los datos, acuso de recibo de mensajes, etc. Los **protocolos de red,** generalmente utilizan un mecanismo llamado **conmutación de paquetes** para enviar y recibir los mensajes en forma de paquetes

Al hablar de **protocolos**, es importante mencionar el modelo **OSI**. Este modelo define un estándar para las comunicaciones entre distintos dispositivos. Siete capas son las que forman partes de este modelo:

- Capa de **aplicación**
- Capa de **presentación**
- Capa de **sesión**
- Capa de **transporte**
- Capa de **red**
- Capa de **datos**
- Capa **física**

Funciones de los protocolos de la capa de aplicación

Tanto el dispositivo **emisor** como el **receptor** utilizan **protocolos de capa de aplicación** durante una comunicación. Para que la comunicación sea exitosa, todos los dispositivos que participan deben comunicarse con el mismo protocolo. Los protocolos de la capa de aplicación deben realizar las siguientes tareas:

- **Establecer reglas consistentes** para el intercambio de datos entre las aplicaciones y servicios cargados en el emisor y receptor.
- **Especificar cómo se estructuran los datos**, y los tipos de mensajes que pueden enviarse los participantes de la comunicación.

Modelo cliente-servidor

En el modelo cliente-servidor, dos dispositivos forman parte de la comunicación. Un dispositivo que solicita información llamado **cliente**, y un dispositivo que provee tal información llamado **servidor**. El cliente inicia la comunicación o intercambio, enviando un pedido (**request**). Este pedido puede contener información requerida por el servidor para generar la respuesta (**response**).

Protocolo HTTP

El protocolo HTTP (HyperText Transfer Protocol) es un **protocolo sin estado (los mensajes son independientes entre si)**, que forma parte de la capa de aplicación y permite la comunicación entre **sistemas distribuidos**. Este protocolo es la base de la web como la conocemos ahora. HTTP permite la **comunicación** entre **clientes** y **servidores** bajo distintas configuraciones de red.

Para distinguir las conexiones, se realiza a través de la asignación de **puertos** a cada una de las aplicaciones

La comunicación entre el servidor y el cliente ocurre a través de un par **request/response**. La **request** o **petición** es la iniciadora de la comunicación, y la realiza el cliente a través de una dirección **URL**

http://www.dominio.com:8080/pagina/archivo.php?a=1&b=2

Protocolo	Servidor o *host*	Puerto	Ruta al recurso	Consulta
http://	***www.dominio.com***	***8080***	***/pagina/archivo.php***	***?a=1&b=2***

Cuando utilizamos el protocolo HTTP, **con especificar la URL no es suficiente**. Además de la URL para "ubicar" el host, es necesario especificar el método que queremos utilizar. El **método** especifica la acción que debe realizar el host en base a la petición del cliente. Los cuatro métodos más comunes son:

Los cuatro métodos más comunes son:
• **GET:** Este método está pensado para obtener un recurso existente del servidor, aunque no es exclusivamente para ello. Cuando se utiliza el método GET para una petición, toda información extra que necesite ser enviada al servidor se codificará en la URL.

• **POST:** Este método está pensado para la creación de un nuevo recurso en el servidor. Suele utilizarse pasando parámetros al servidor, pero estos parámetros no se codifican como parte de la URL, sino que viajan en el cuerpo de la petición. Esto hace que sea un método más seguro, **ya que la información no es visible directamente para el usuario.**

• **PUT:** Este método se utiliza para actualizar un recurso. Suele ir acompañado de información al igual que GET y POST.

• **DELETE:** Se utiliza para eliminar recursos existentes.

Respuestas y códigos de status

Una vez que el cliente realiza la petición (**request**), el servidor realiza las operaciones correspondientes y devuelve una respuesta (**response**). Las respuestas del servidor llevan consigo un código que representa **cómo se completó la operación**, y de qué manera la misma **debe ser interpretada por el cliente**.

Los códigos 2xx indican que la petición fue procesada correctamente por parte del servidor. El código más común cuando una petición se procesa correctamente es el código **200**.

• **202.** La petición de un recurso fue aceptada, pero el mismo puede no estar incluido en la respuesta.
• **204.** Representa una respuesta que no tienen ningún contenido.
• **205.** Indica al cliente que vuelva a mostrar el contenido de cero.
• **206.** Indica que la respuesta contiene solo una parte de la respuesta

Los códigos 3xx indican al cliente que debe realizar alguna acción extra. El caso más común es que el servidor indique al cliente que debe redireccionarse a una dirección distinta.

• **301.** El recurso fue movido permanentemente (cambia su URL).
• **303.** El recurso fue movido temporalmente y el usuario debe redirigirse. Como parte de la respuesta, se envía la URL temporal.
• **304.** El servidor determina que el recurso requerido por el cliente no se ha modificado desde la última vez que lo consultó, por lo tanto, indica que debe utilizar la copia almacenada en caché.

Los códigos 4xx son indican al cliente que hay un error en la petición. Esto puede suceder cuando se solicita un recurso que no existe o hay algún problema en la petición misma. Algunos de los códigos 4xx más comunes son:

• **400.** Indica que la petición está mal realizada (se conoce como 400 Bad Request).
• **401.** Indica la necesidad de una autenticación del cliente.
• **403.** Indica que el servidor niega al cliente el acceso al recurso solicitado.
• **404.** Indica que el recurso solicitado no fue encontrado. **(404 not found)**
• **405.** Indica que el método utilizado en la petición (GET, POST, etc.) no está permitido, o el servidor no lo soporta.

Los códigos 5xx son utilizados para indicar un fallo en el servidor durante el procesamiento de la petición.
• **500.** Indica un error interno en el servidor (Internal server error)
• **501.** Indica que el servidor todavía no soporta la operación pedida por el cliente (Not implemented)
• **503.** Este código se devuelve cuando existe un problema interno en el servidor, o cuando el mismo está sobrecargado. El servidor directamente no devuelve ninguna respuesta. Se produce un **timeout**.

Protocolo FTP

El protocolo FTP **(File Transfer Protocol)** es uno más dentro de la capa de aplicación, y se utiliza para la transferencia de archivos a través de la red. Un servidor que ofrezca un servicio FTP, nos permitirá **acceder a directorios y archivos en un determinado directorio**, o la raíz de directorios completa, dependiendo de los permisos que tengamos. Existen múltiples maneras de conectarse a un servidor FTP.

• **Conexión desde un navegador:** Para conectarse a un servidor FTP a través de un navegador, tendremos que utilizar una URL, al igual que en HTTP. La URL debe tener la siguiente forma: **ftp**://usuario@dominioftp.gov/

• **Conexión desde la línea de comandos:** En la mayoría de los sistemas operativos existe un comando para establecer conexiones FTP.

• **Conexión a través de una interfaz gráfica** Generalmente estas aplicaciones además de permitirnos navegar por FTP, permiten otras operaciones como modificación de los archivos en el servidor, copia, eliminación, etc. Si bien esto es posible realizarlo íntegramente desde la línea de comandos, una interfaz gráfica nos simplificará las tareas.

Protocolo SSH

El protocolo SSH **(Secure Shell),** es un método para iniciar una sesión segura, en una computadora remota, utilizando distintos métodos como autenticación con **passwords**, llaves SSH, etc. Una vez iniciada la sesión, la información transmitida desde el servidor hacia el cliente, y viceversa. Se utilizan para:

• Proveer acceso a usuarios y procesos automáticos a un servidor remoto.
• Transferencia de archivos. Si bien se puede utilizar FTP para ello, SSH ofrece una alternativa más segura.
• Ejecutar comandos de manera remota.

*En Linux se puede utilizar el comando **ssh** especificando el servidor al que se desea conectar*

Redes P2P

Las redes P2P, son aquellas que **conectan múltiples clientes entre sí** (también llamados pares o peers), **de una manera determinada y con un fin específico**.

Se utiliza generalmente para compartir archivos; aplicaciones, archivos de video, imágenes, documentos, etc.

A diferencia de las redes tradicionales cliente-servidor, donde hay siempre un servidor (o múltiples servidores, pero que sirven para un mismo propósito), y múltiples clientes, en las redes P2P, **cada nodo (dispositivo conectado a la red), tiene el rol de cliente y servidor al mismo tiempo.**

Esto hace que cada cliente pueda estar consumiendo recursos de otro cliente o que esté sirviendo recursos para un cliente remoto.