

SISTEMAS OPERATIVOS I



BIENVENIDA

Bienvenido(a) a la asignatura *Sistemas Operativos I*, con la cual aprenderás todo lo necesario para aprender a utilizar el sistema operativo Linux. Además, conocerás su composición, sus principales conceptos (como *Kernel* y *Shell*) y sus comandos básicos (como entradas y salidas, etcétera).

A diferencia de Windows y MacOS (los cuales son sistemas de código cerrado, diseñados para uso comercial y personal), Linux es un *Kernel* de código abierto, el cual admite una variedad de sistemas de todo tipo.

LIBROS RECOMENDADOS



Clinton., D. (2018) Linux in Action. Manning.

Casto., J. D. (2016) Introducing Linux Distros: Choose the right Linux distribution for your needs. Apress.

UNIDAD 1

INTRODUCCIÓN A LINUX

```
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86
```





1.1

TEMARIO

ESTRUCTURA DEL
SISTEMA OPERATIVO

1.2

COMANDOS
BÁSICOS



INTRODUCCIÓN

La asignatura *Sistemas Operativos I* tiene como principal objetivo que comprendas el uso de Linux. Lo anterior con el fin de diferenciarlo de Windows, principalmente, e identifiques sus ventajas frente a este último.

En esta primera unidad aprenderás cómo se conforma la estructura del sistema operativo de Linux. Además, conocerás los comandos básicos que se utilizan en el *Shell*.



COMPETENCIAS A DESARROLLAR



- El alumno conocerá y comprenderá los comandos básicos de *Shell Linux*, así como su uso básico.

- El alumno comprenderá de manera clara la estructura sobre la que se construye Linux.

ESTRUCTURA DEL SISTEMA OPERATIVO

Para iniciar... ¿Qué es Linux?

Al igual que Windows, iOS y Mac OS, Linux es un sistema operativo*. Se trata de una de las plataformas más populares del planeta. De hecho, Android funciona con el sistema operativo Linux.

*Un sistema operativo es un software que administra todos los recursos de hardware asociados con tu computadora de escritorio o laptop.





Además, Linux es tanto un ***Kernel*** como un sistema operativo que se ejecuta sobre él. Esto depende del contexto en el que encuentre la referencia.

El ***Kernel de Linux*** fue creado en 1991 por Linus Torvalds. Hoy en día, es mantenido por una comunidad mundial de desarrolladores, entre los que se incluyen programadores individuales, y empresas como IBM, HP y Hitachi. Todos ellos coordinados por el mismo Linus, ahora un desarrollador de renombre mundial.

El sistema operativo Linux consta de la siguiente estructura:

- Bootloader
- Kernel
- Init System
- Daemons

A continuación, analizaremos cada uno de estos conceptos a detalle.

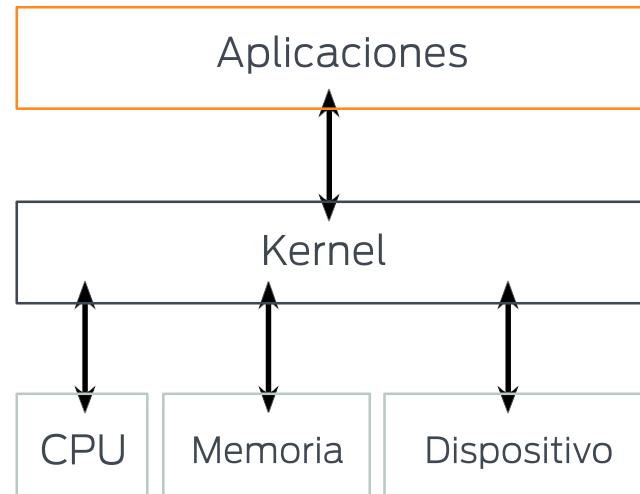


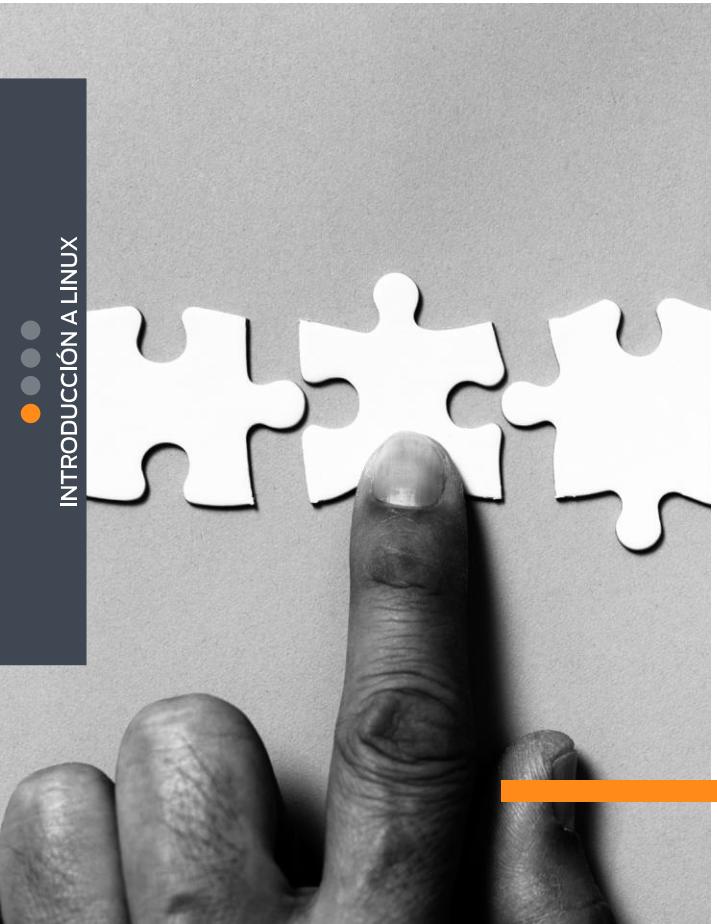
- **Bootloader.** También llamado administrador de arranque, es un pequeño programa que coloca el Sistema Operativo (SO) de una computadora en la memoria. Así, cuando se enciende o se reinicia una computadora, el Sistema Básico De Entrada/Salida (BIOS) realiza algunas pruebas iniciales y luego transfiere el control al Registro de Arranque Maestro (MBR), donde reside el cargador de arranque.



- **Kernel.** Es un componente central de un sistema operativo, el cual se encarga de administrar las operaciones de la computadora y el hardware. Básicamente gestiona operaciones de memoria y tiempo de CPU. Es el componente central de un sistema operativo.

Kernel actúa como un puente entre las aplicaciones y el procesamiento de datos realizado a nivel de hardware. Esto mediante la comunicación entre procesos y las llamadas al sistema.





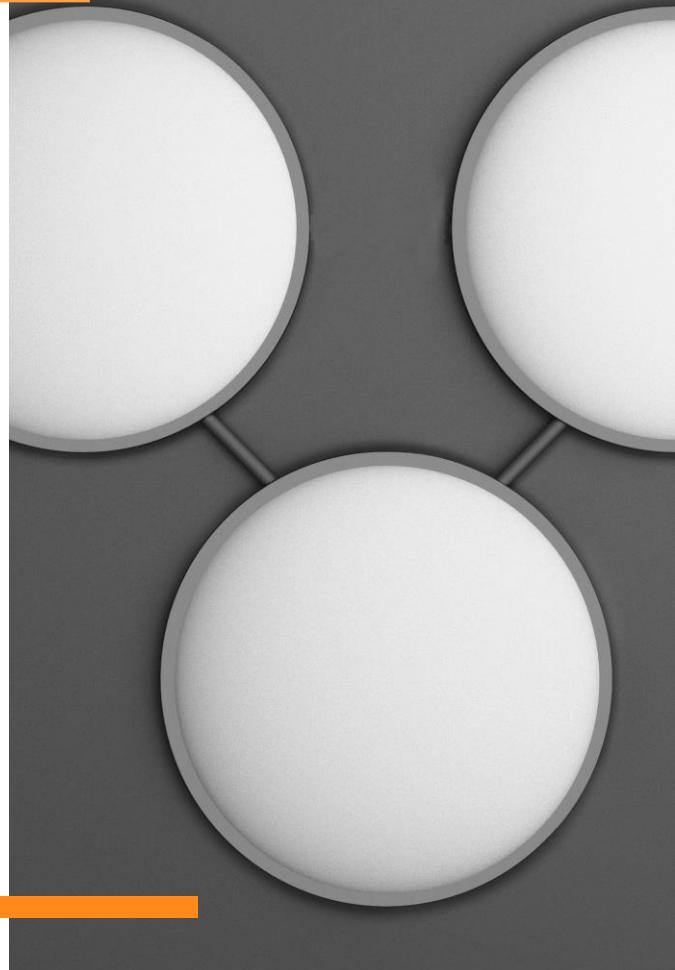
Además, el *kernel* se carga primero en la memoria cuando se inicializa un sistema operativo, y permanece en la misma hasta que este se apaga nuevamente. Por lo tanto, es responsable de varias tareas, como la gestión del disco, la gestión de tareas y la gestión de la memoria.

Además, el *kernel* decide qué proceso debe asignarse al procesador para ejecutar, así como qué proceso debe mantenerse en la memoria principal para ejecutar.

El *kernel*, básicamente, actúa como una **interfaz entre las aplicaciones de usuario y el hardware**.

El objetivo del *kernel* (o núcleo) es gestionar la comunicación entre el software, es decir, las aplicaciones a nivel de usuario; y el hardware, es decir, la CPU y la memoria del disco. Por ello, **el kernel permite:**

- Establecer comunicación entre la aplicación a nivel de usuario y el hardware.
- Decidir el estado de los procesos entrantes.
- Controlar la gestión del disco, la gestión de la memoria y la gestión de tareas.





- **Init Systems.** Es el primer proceso que se inicia en Fedora después de que se inicia el *kernel*. De hecho, el sistema *init* siempre obtiene el ID de proceso (**PID**) “1” en un sistema.

Por su parte, el *kernel* de Linux siempre ejecuta el proceso *init* después de que el BIOS y el gestor de arranque (GRUB) completan las primeras etapas del arranque.

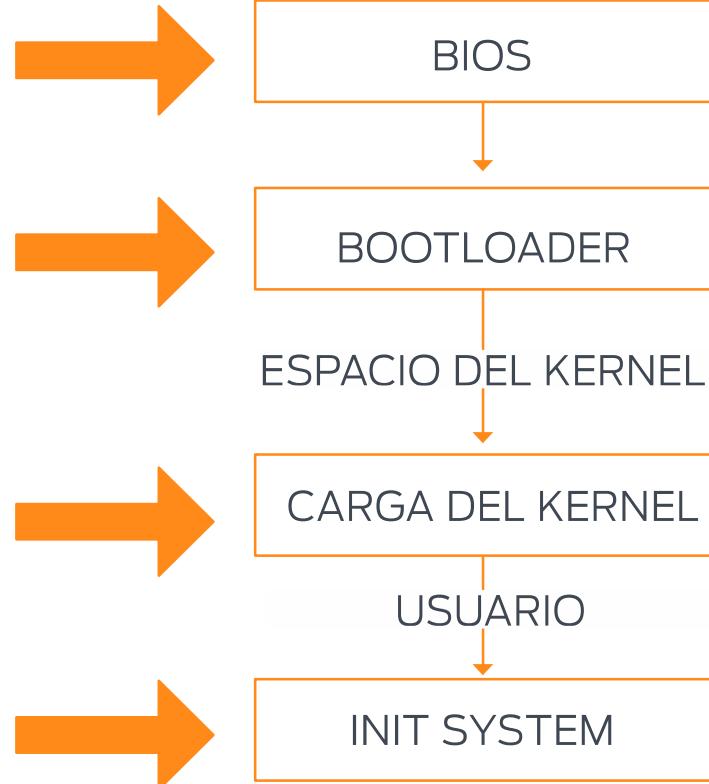
ESTRUCTURA DEL SISTEMA OPERATIVO

Este es el GRUB o LILO donde se elige el SO que se desea iniciar.

El Kernel del SO carga sus extensiones, controladores de dispositivos.

El primer proceso: ejecuta procesos en el arranque y administra todos los procesos hasta que se apaga.

EN EL ENCENDIDO



- **Daemons.** El sistema *init* debe iniciar todos los *daemons*, que son todos los procesos en segundo plano y servicios necesarios para que el sistema operativo funcione.

Estos procesos que continúan ejecutándose en segundo plano después de que se inician se denominan *daemons*. Estos administran muchas partes de tu sistema, por ejemplo: el registro de información y la vigilancia de los dispositivos que inserta o elimina.





ESTRUCTURA DEL SISTEMA OPERATIVO

Shell. Proporciona una interfaz para el sistema Unix. Recopila información del usuario y ejecuta programas basados en esa entrada. Además, cuando un programa termina de ejecutarse, muestra la salida de este.

Shell es un entorno en el que podemos ejecutar comandos, programas y *scripts*. Hay diferentes tipos de *Shell*; al igual que hay diferentes tipos de sistemas operativos. Adicionalmente, cada tipo de *Shell* tiene su propio conjunto de funciones y comandos reconocidos.



VIDEO

Te invitamos a ver el siguiente video:



COMANDOS BÁSICOS

Como vimos anteriormente, un *Shell* es un programa que recibe comandos del usuario y se los da al sistema operativo para que los procese; luego, muestra el resultado.

El *Shell* de Linux es su parte principal. Sus distribuciones vienen en GUI (interfaz gráfica de usuario), pero básicamente, Linux tiene una CLI (interfaz de línea de comandos).

En este contexto, veamos los **comandos básicos que podemos ejecutar en *Shell*.**



T ► **Pwd.** Para saber en qué directorio te encuentras puedes usar este comando. Sirve para darnos la ruta absoluta, es decir, la que comienza desde la raíz. Cabe mencionar que la raíz es la base del sistema de archivos de Linux. Esta se denota con una barra diagonal “/”. El directorio de usuarios suele ser algo así como: **/home/username**

```
usuario:~$ pdw/home/nayso
```

2

- **ls.** Este comando nos ayuda a saber qué archivos hay en el directorio en el que se encuentra. Además, se pueden ver todos los archivos ocultos usando el comando **ls -a**

```
Desktop           itsuserguide.desktop  reset-settings      VCD_Copy
Documents        Music                  School_Resources   Videos
Download         Pictures               Students_Works_10
examples.desktop Public                Templates
GplatesProject   Qgis Projects        TuxPaint-Pictures
```

3

COMANDOS BÁSICOS

- ▶ **cd.** Este comando sirve para ir a un directorio. Por ejemplo: si estás en la carpeta de inicio y deseas ir a la carpeta de descargas, puedes escribir **cd Descargas**. Cabe señalar que este comando distingue entre mayúsculas y minúsculas, por lo que debes escribir el nombre de la carpeta exactamente como está.

```
usuario:~$ cd Downloads  
usuario:~/Downloads$ cd
```



- ▶ **mkdir y rmdir.** Por una parte, el comando **mkdir** se usa cuando se necesita crear una carpeta o un directorio. Por ejemplo, si quieres crear un directorio llamado **DIY**, puedes escribir lo siguiente: **mkdir DIY**.

Por otro lado, **rmdir** se utiliza para eliminar un directorio. Cabe mencionar que solo se puede usar para eliminar un directorio vacío.

```
usuario:~/Desktop$ ls
usuario:~/Desktop$ mkdir DIY
usuario:~/Desktop$ ls
DIY
usuario:~/Desktop$ rmdir DIY
usuario:~/Desktop$ ls
usuario:~/Desktop$
```

5

COMANDOS BÁSICOS

- **rm.** Este comando sirve para eliminar archivos y directorios. Para eliminar solo el directorio, se utiliza **rm -r**, ya que se eliminan tanto la carpeta como los archivos que contiene cuando se usa solo el comando **rm**.

```
usuario:~/Desktop$ rm newer.py
usuario:~/Desktop$ ls
New Folder
usuario:~/Desktop$ rm -r Ner\ Folder
usuario:~/Desktop$ ls
usuario:~/Desktop$
```

6

- **touch.** Este comando se utiliza para crear un archivo. Este puede ser de cualquier tipo, desde un archivo *txt* vacío hasta un archivo *zip*. Por ejemplo: **touch new.txt**

```
usuario:~/Desktop$ ls
usuario:~/Desktop$ touch new.txt
usuario:~/Desktop$ ls
new.txt
```

7

- ▶ **man & --help.** Por un lado, el comando **man** se utiliza para saber más sobre un comando y cómo usarlo. Este muestra las páginas del manual del comando. Por ejemplo: **man cd** muestra las páginas del manual del comando **cd**.

Por su parte, escribir el nombre del comando y el argumento **help** muestra de qué manera se puede usar el comando.

TOUCH(1)**User Commands****TOUCH(1)****Name****touch - change file timestamps****SYNOPSIS****touch [OPTION]... FILE...****Description****Update the access and notification times for each FILE to the current time.****A FILE argument that does not exist is created empty, unless -c or -h is supplied.****A FILE argument string of - ls handled specially and causes touch to change the times of the file associated with standard output****Mandatory arguments to long options are mandatory for short options too.****-a change only the access time**

8

COMANDOS BÁSICOS

- **cp.** Este comando se utiliza para copiar archivos a través de la línea de comando. Para ello, toma dos argumentos: el primero es la ubicación del archivo que se va a copiar, y el segundo es la dirección a donde se va a copiar.

```
usuario:~/Desktop$ ls /home/usuario/Music
usuario:~/Desktop$ ls cp new.txt  /home/usuario/Music
usuario:~/Desktop$ ls /home/usuario/Music
new.txt
```



- ▶ **mv.** Este comando es utilizado para mover archivos a través de la línea de comandos. Además, también se puede usar para cambiar el nombre de un archivo. Por ejemplo, si queremos renombrar el archivo **text** a **new**, podemos usar **mv text new**.

```
usuario:~/Desktop$ ls  
new.txt  
usuario:~/Desktop$ mv new.txt newer.txt  
usuario:~/Desktop$ ls  
newer.txt
```

10

- ▶ **Locate.** Este comando se utiliza para encontrar un archivo en un sistema Linux. Es útil cuando no se conoce el lugar donde se guarda un archivo, o bien, su nombre real.

Por lo tanto, si desea encontrar, por ejemplo, un archivo que tenga la palabra **Hello**, se listarán todos los archivos en tu sistema Linux que contengan dicha palabra. Si recuerdas solo dos palabras, puedes separarlas con un asterisco.

```
usuario:~$ locate newer.txt  
/home/usuario/Desktop/newer.txt  
usuario:~$ locate *DIY*Hacking*  
/home/usuario/DIY Hacking
```

11

- **ping.** Este comando nos ayuda a verificar la conexión a un servidor, ya que mide el tiempo de ida y vuelta entre este y nuestro equipo y brinda los detalles al respecto. Por lo tanto, este comando nos ayuda a verificar la conexión a Internet.

```
usuario:~/Desktop$ ping google.com
PING google.com (172.217.26.206) 56(84) bytes of data.
64 bytes from google.com (172.217.26.206) icmp_seq=1 ttl=56 time=51.2 ms
64 bytes from google.com (172.217.26.206) icmp_seq=2 ttl=56 time=51.2 ms
64 bytes from google.com (172.217.26.206) icmp_seq=3 ttl=56 time=51.2 ms
^C
- - - google.com ping statistics - - -
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 47.959/49.388/51.299/1.417 ms
```

12

- ▶ **hostname.** Este comando te ayuda a conocer tu nombre en el *host* o red. Lo que hace es, básicamente, mostrar el nombre de *host* y dirección IP. Si se escribe **nombre de host**, este comando nos da la salida. Por otro lado, si se escribe **nombre de host -I**, se muestra la dirección IP en tu red.

```
usuario:~/Desktop$ hostname User  
usuario:~/Desktop$ hostname -I 192.168.1.36
```

13

- ▶ **uname.** Este comando sirve para mostrar la información sobre el sistema que ejecuta su distribución de Linux. Usando el comando **uname -a** imprime la mayor parte de la información sobre el sistema. Esto imprime la fecha de lanzamiento del *kernel*, la versión, el tipo de procesador, etc.

```
usuario:~$ uname -a
Linux usuario 4.4.0-22.generic #40~14.04.1-Ubuntu SMP Fri Jul 15 17:27:18 UT
C 2022 i686 i686 i686 GNU/Linux
```



LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 1

- Morelo, B. D. (2018). *What Is a Boot Loader?* MakerPro.

Capítulo 2

- Unix / Linux. (2009). *What is Shells?* Tutorials Point.



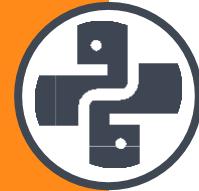


Entorno de trabajo.

Participa en el foro enviando imágenes que demuestren que ya tienes acceso a las siguientes herramientas en su versión de prueba:

- Sistema operativo Linux

Presiona el botón para participar en el foro.

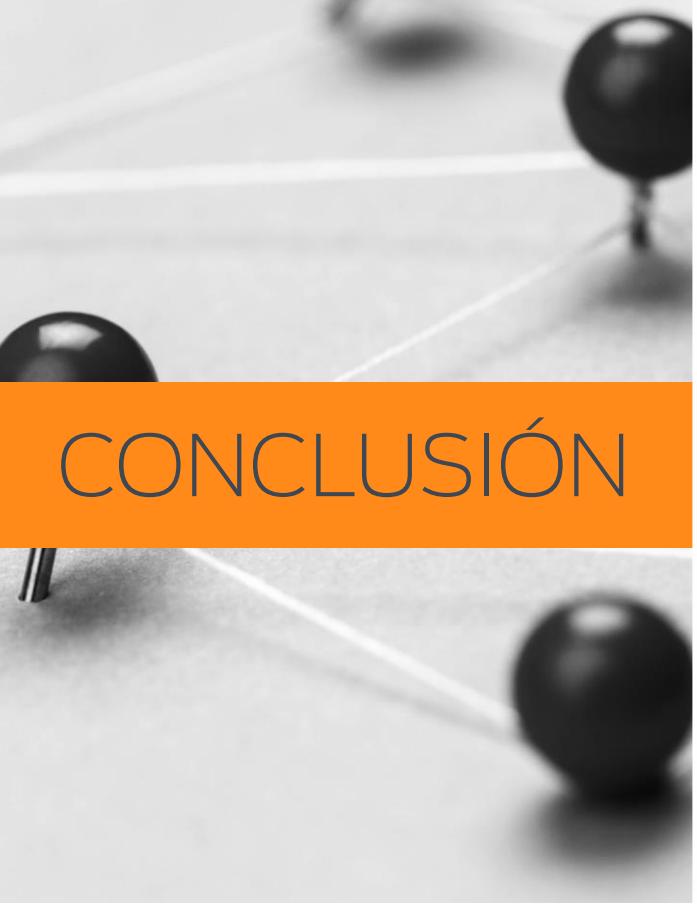


FORO UNIDAD 1



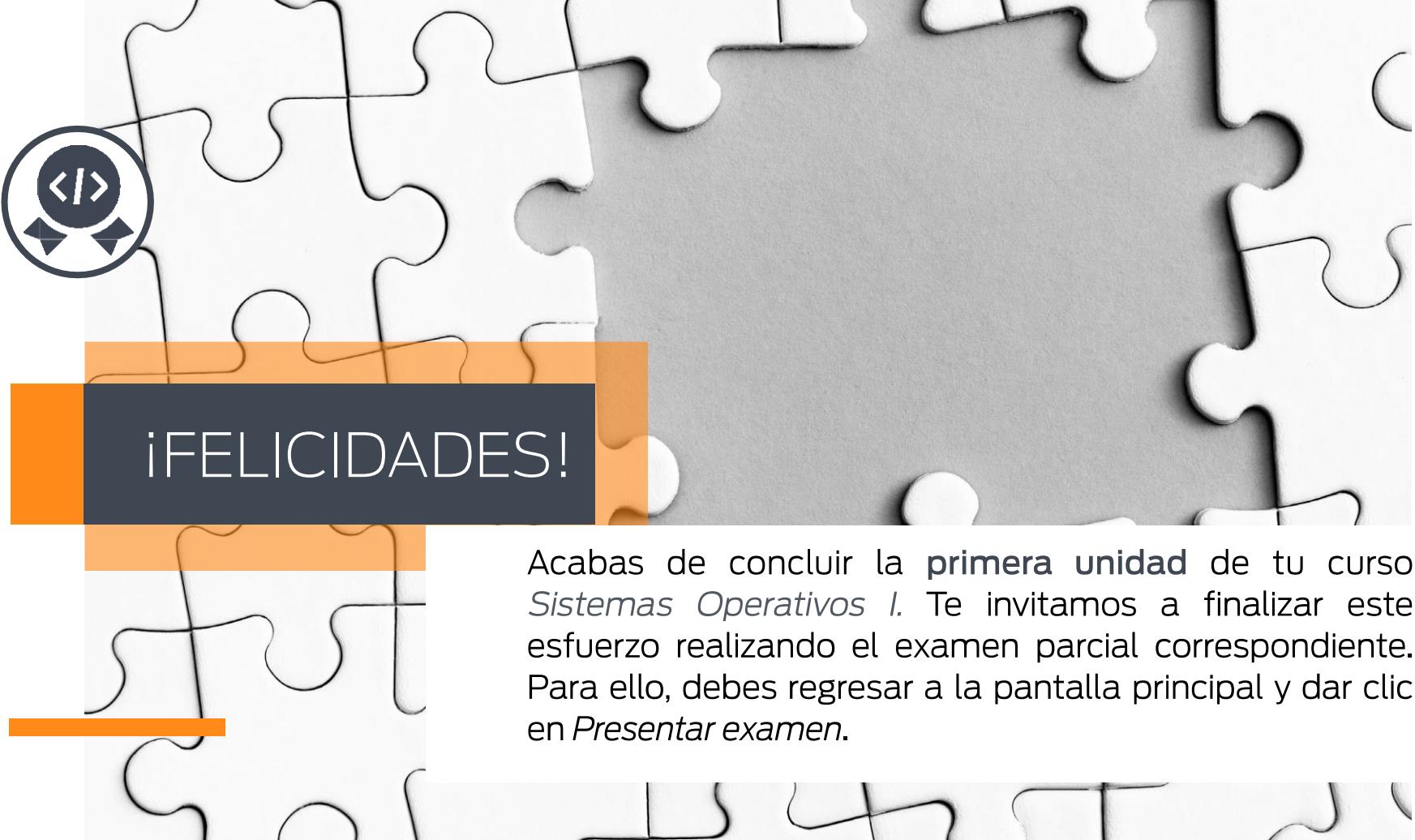


CONCLUSIÓN



Linux ha existido durante mucho tiempo y, lamentablemente, ha sido ignorado para el uso de escritorio convencional durante la mayor parte de su existencia. Sin embargo, en estos últimos años ha tomado mucha relevancia en el mundo tecnológico. Hace dos años, muchos software no sacaban una versión para Linux, pero sí para Windows o MacOS. No obstante, es obligatorio sacar la compatibilidad con Linux.

El movimiento de código abierto ha crecido hasta convertirse en una fuerza importante en el entorno informático actual. Linux se está volviendo cada vez más popular como sistema operativo para servidores, y gradualmente le quita cuota de mercado a Windows.



Acabas de concluir la **primera unidad** de tu curso *Sistemas Operativos I*. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

UNIDAD 2

COMPONENTES PRINCIPALES



```
code.txt
1  while choice!="q":
2      print ("1-----")
3      print ("1 Score\n" + "2 Settings\n" + "3 Game\n" + "q Quit")
4
5      choice = input("Your Choice: ")
6
7      ## Prints the score of the previous game
8      if choice == "1":
9          print ("-----")
10         myfile.printScores()
11         print ("-----")
12
13      ## The player can choose the level of the PC
14      if choice == "2":
15          print ("-----")
16          print ("1 Easy\n" + "2 Normal\n" + "3 Hard")
17
18
19
20
21
22
23
24
25
26
27
28
```



2.1

TEMARIO

SHELL

2.2

KERNEL



INTRODUCCIÓN



En esta segunda unidad conocerás los diferentes tipos que existen tanto de *Shell* como de *Kernel*. Además, comprenderás sus principales diferencias y particularidades. Aunado a lo anterior, tendrás en cuenta cuáles son los tipos de *Kernel* y *Shell* que utiliza Linux, así como su arquitectura.



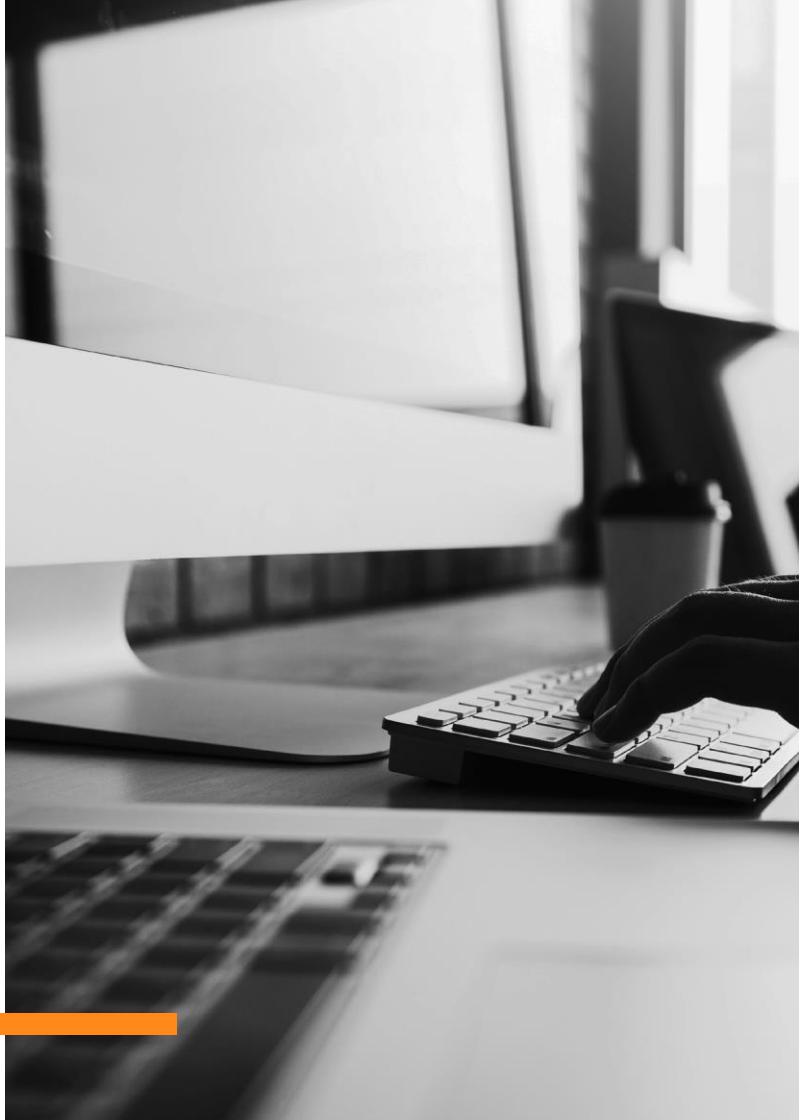
COMPETENCIAS A DESARROLLAR

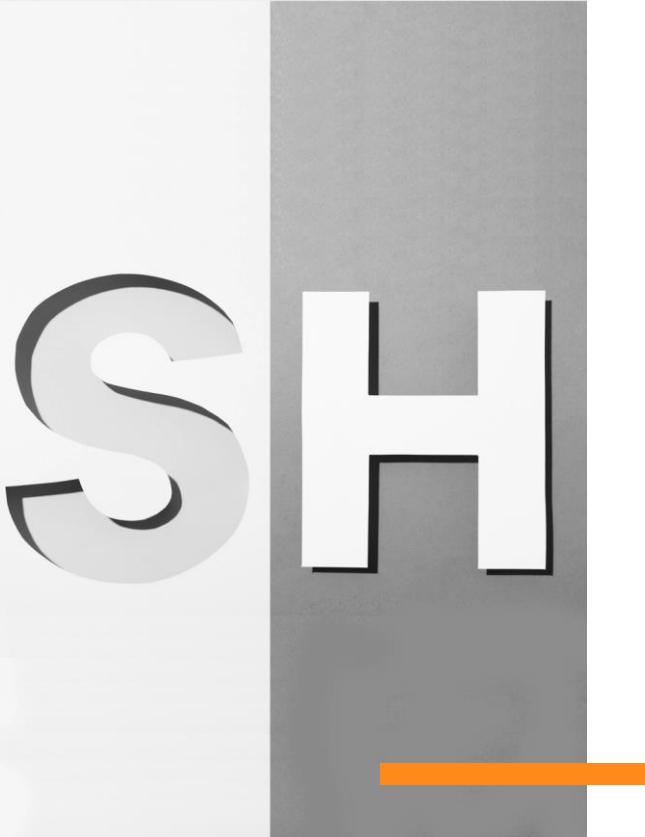
- El alumno será capaz de conocer las arquitecturas tanto del *Kernel* como del *Shell* de Linux para comprender mejor su funcionamiento.

- El alumno conocerá y comprenderá los diferentes tipos de *Kernel* y *Shell* existentes.

SHELL

Shell. Se trata de un programa que toma comandos del teclado para dárselos a un sistema operativo, el cual los pueda ejecutar. Anteriormente, el *Shell* era la única interfaz de usuario disponible en un sistema similar a Unix como Linux. No obstante, en la actualidad tenemos para ello Interfaces Gráficas de Usuario (GUI), así como Interfaces de Línea de Comandos (CLI) como el *Shell*.





Ahora bien, en la mayoría de los sistemas Linux existe un programa llamado ***bash*** (*Bourne Again Shell*). Se trata de una versión mejorada del programa *Shell* original **sh**.

Además de *bash* existen otros programas de *Shell* disponibles para sistemas Linux, por ejemplo:

- **ksh**
- **tcsh**
- **zsh**

El *Shell* de Unix proporciona una interfaz que permite al usuario interactuar con el sistema operativo mediante la ejecución de comandos. Sin embargo, un *Shell* también es un **lenguaje de programación** que puede realizar las siguientes funciones:

- Construcciones para control de flujo
- Alternancia
- Bucles
- Condicionales
- Operaciones matemáticas básicas
- Funciones con nombre, entre otras.



Los *shells* se pueden usar de **forma interactiva** (una terminal o un emulador de terminal como *xterm*); y de **forma no interactiva** (leyendo comandos desde un archivo).

La mayoría de los *Shells* modernos proporcionan edición de línea de comandos, la cual se puede manipular usando comandos de tipo **emacs** o **vi** mientras se ingresa; así como varias formas de un historial de comandos guardado.



Tipos de *Shells* disponibles

Cada *Shell* tiene propiedades únicas, que los hacen altamente eficientes para un tipo de uso específico. Algunos de ellos son:

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell
- The Z Shell (zsh)



- **The Bourne Shell (sh).** Se trata del *shell* UNIX original. Además, es el que más se usa para la programación de *Shell*. Sin embargo, carece de funciones para uso interactivo, así como de manejo de expresiones aritméticas y lógicas incorporadas.

Sin embargo, este es el *Shell* predeterminado del SO Solaris.

El nombre de la ruta completa del comando es `/bin/sh` y `/sbin/sh`

El indicador predeterminado de usuario **no raíz** es `$`

El indicador predeterminado del usuario **raíz** es `#`

- **The C Shell (csh).** Fue desarrollado para incluir funciones de programación útiles, como soporte integrado para operaciones aritméticas y una sintaxis similar al lenguaje de programación C.

Además, incorporó el historial de comandos que faltaba en diferentes tipos de *Shells* en Linux como el *shell Bourne*. Otra característica destacada de un Shell C son los alias.

El nombre de la ruta completa del comando es **/bin/csh**

El indicador predeterminado del **usuario no raíz** es **hostname %**

El aviso predeterminado del **usuario raíz** es **hostname #**



SHELL

- **The Korn Shell (ksh).** Es esencialmente un superconjunto del caparazón *Bourne*, ya que, además de admitir todo lo que sería compatible con este, proporciona a los usuarios nuevas funcionalidades. Permite soporte incorporado para operaciones aritméticas, mientras ofrece características interactivas que son similares al *Shell C*.

El *Shell Korn* ejecuta *scripts* creados para el *Shell Bourne*, al tiempo que ofrece una manipulación de cadenas, matrices y funciones similares al lenguaje de programación C.

También admite *scripts* que se escribieron para el *shell C*. Además, es más rápido que la mayoría de los diferentes tipos de *Shells* en Linux, incluido el *Shell C*.

El nombre de la ruta completa del comando es **/bin/ksh**

El indicador predeterminado de **usuario no raíz** es **\$**

El indicador predeterminado del **usuario raíz** es **#**

- **The GNU Bourne-Again Shell.** Conocido popularmente como *Shell Bash*, fue diseñado para ser compatible con el *Shell Bourne*, ya que incorpora características útiles de diferentes tipos de *Shells* en Linux.

Además, permite recuperar automáticamente los comandos utilizados anteriormente y editarlos con la ayuda de las teclas de flecha, a diferencia del *Shell Bourne*.

El nombre de la ruta completa del comando es **/bin/bash**

El aviso predeterminado para un **usuario no raíz** es **bash-x.xx\$**

El aviso predeterminado del **usuario raíz** es **bash-x.xx#**

Bash se usa para cualquier aplicación informática que requiera precisión al trabajar con archivos y datos, especialmente cuando es necesario buscar, clasificar, manipular o procesar grandes cantidades de archivos o datos de alguna manera.

Además, *Bash* se usa comúnmente de forma interactiva, pero también se puede usar para escribir *scripts* de *Shell*. Casi cualquier tarea informática se puede automatizar mediante un *script Bash*.



El comando **ls** tiene numerosos parámetros, los cuales modifican cómo se muestran los resultados. Algunos parámetros de uso frecuente con el comando **ls** incluyen:

Argumentos de la línea de comandos de ls	Objetivo
-l	Utiliza un formato de lista más largo y detallado para incluir los permisos de archivo, el propietario del archivo, el grupo, el tamaño y la fecha/hora de creación.
-a	Enumera todos los archivos y subdirectorios, incluso aquellos que normalmente están destinados a ocultarse.
-s	Muestra el tamaño de cada archivo.
-h	Muestra los tamaños de archivos y subdirectorios en formato legible por humanos usando K, M, G, etc. para indicar kilobytes, megabytes y gigabytes.
-R	Usados en conjunto, estos parámetros brindan al usuario una idea mucho más clara de qué archivos y subdirectorios hay en un directorio, cuándo se modificaron por última vez y quién lo hizo.

- **The Z Shell (zsh).** Es una extensión de *sh Shell*, pero con muchas mejoras para la personalización. Si lo que se busca es un *Shell* moderno que tenga todas las características, este es el *Shell* que se debe elegir.

Permite generar nombres de archivo basados en condiciones dadas.

Permite complementos y soporte de temas.

Facilita el índice de funciones integradas.

Permite la finalización de comandos.

A continuación, te presentamos una breve comparación de los **5 tipos de Shell y sus propiedades**:

Shell	Nombre completo de la ruta	Prompt para el usuario raíz	Solicitud de usuario no raíz
Bourne shell (sh)	/bin/sh and /sbin/sh	#	\$
C shell (csh)	in/csh	#	%
Korn shell (ksh)	/bin/ksh	#	\$
GNU Bourne - Again shell (bash)	/bin/bash	bash-VersionNumber#	bash-VersionNumber\$
Z Shell (zsh)	/bin/zsh	<hostname>#	<hostname>%

Entorno de usuario

Shell ofrece un entorno de usuario, el cual puede personalizarse mediante archivos de inicialización, los cuales tienen **configuraciones** para las características del entorno del usuario. Por ejemplo:

- Rutas de búsqueda para encontrar comandos.
- Permisos predeterminados en archivos nuevos.
- Valores de variables que utilizan otros programas.
- Valores que puedes personalizar.

```
nit@nit-MS-7B85:~$ MYVAR="xyz" &&
echo &MYVAR
xyz
nit@nit-MS-7B85:~$
```

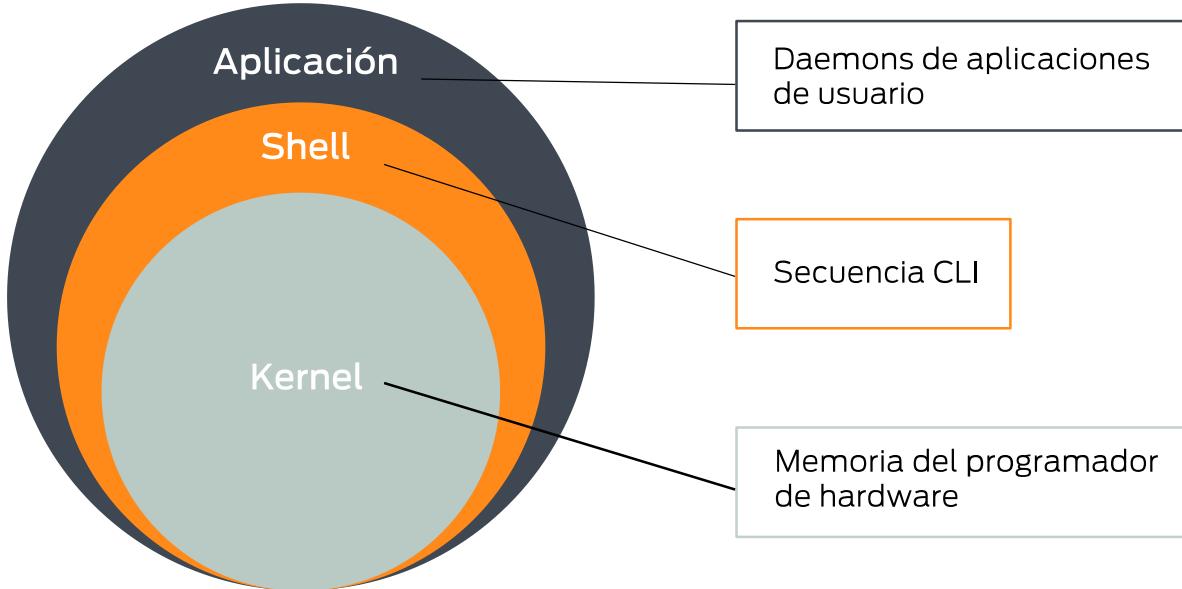


VIDEO

Te invitamos a ver el siguiente video:



KERNEL



Kernel. Representa el nivel más bajo de software, fácilmente reemplazable, que interactúa con el hardware de tu computadora. Es responsable de interconectar todas las aplicaciones que se ejecutan en el modo de usuario hasta el hardware físico. Además, permite que los procesos (servidores) obtengan información entre sí mediante la Comunicación entre Procesos (IPC).

Tipos de Kernel

En general, la mayoría de los núcleos se clasifican en uno de estos **3 tipos**:

- Micronúcleo
- Monolítico
- Híbrido

Cabe señalar que Linux es un *Kernel* monolítico; mientras que OS X y Windows 7 se caracterizan por ser híbridos.





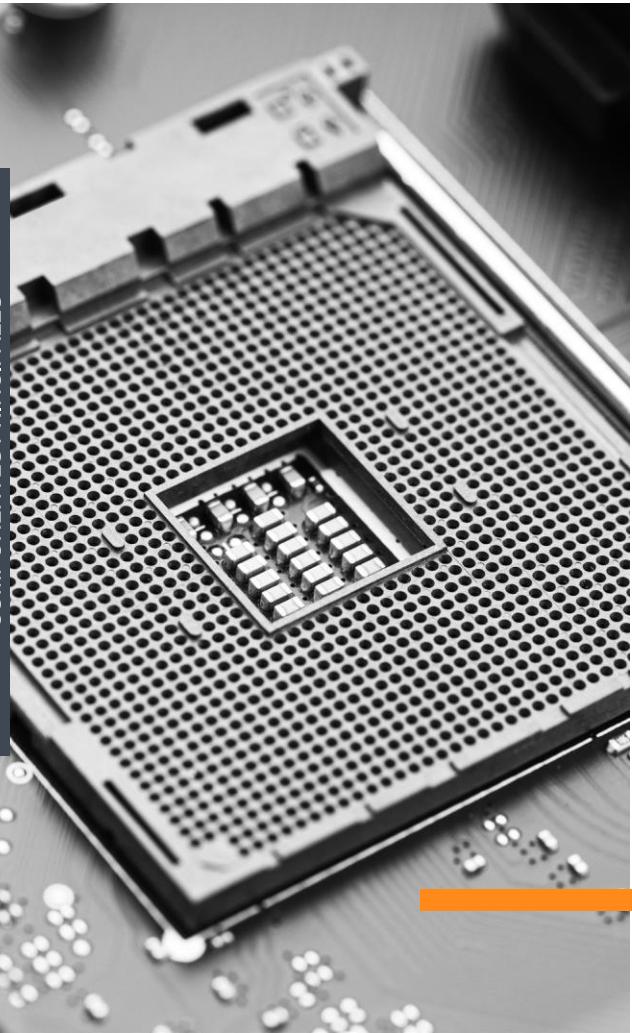
KERNEL

- **Micronúcleo.** Se enfoca en administrar solo lo que tiene que hacer, es decir: CPU, memoria e IPC. Por lo tanto, todo lo demás en una computadora puede verse como un accesorio, y puede manejarse en modo de usuario.

Los micronúcleos tienen la ventaja de la portabilidad, porque no tienen que preocuparse si cambia la tarjeta de video, o incluso el sistema operativo, siempre y cuando este aún intente acceder al hardware de la misma manera.

Además, los micronúcleos también ocupan muy poco espacio, tanto para la memoria como para el espacio de instalación. Asimismo, tienden a ser más seguros, ya que solo se ejecutan procesos específicos en el modo de usuario, el cual no tiene los altos permisos que tiene el modo de supervisor.





- **Monolítico.** Es lo opuesto a los *microkernel*, porque abarca no solo la CPU, la memoria y el IPC, sino que también incluye controladores de dispositivos, administración del sistema de archivos y llamadas al servidor del sistema.

En este sentido, el *kernel* monolítico tiende a ser mejor para acceder al hardware y realizar múltiples tareas. Esto porque si un programa necesita sacar información de la memoria u otro proceso en ejecución, tiene una línea más directa para acceder y no tiene que esperar en una cola para hacer las cosas.

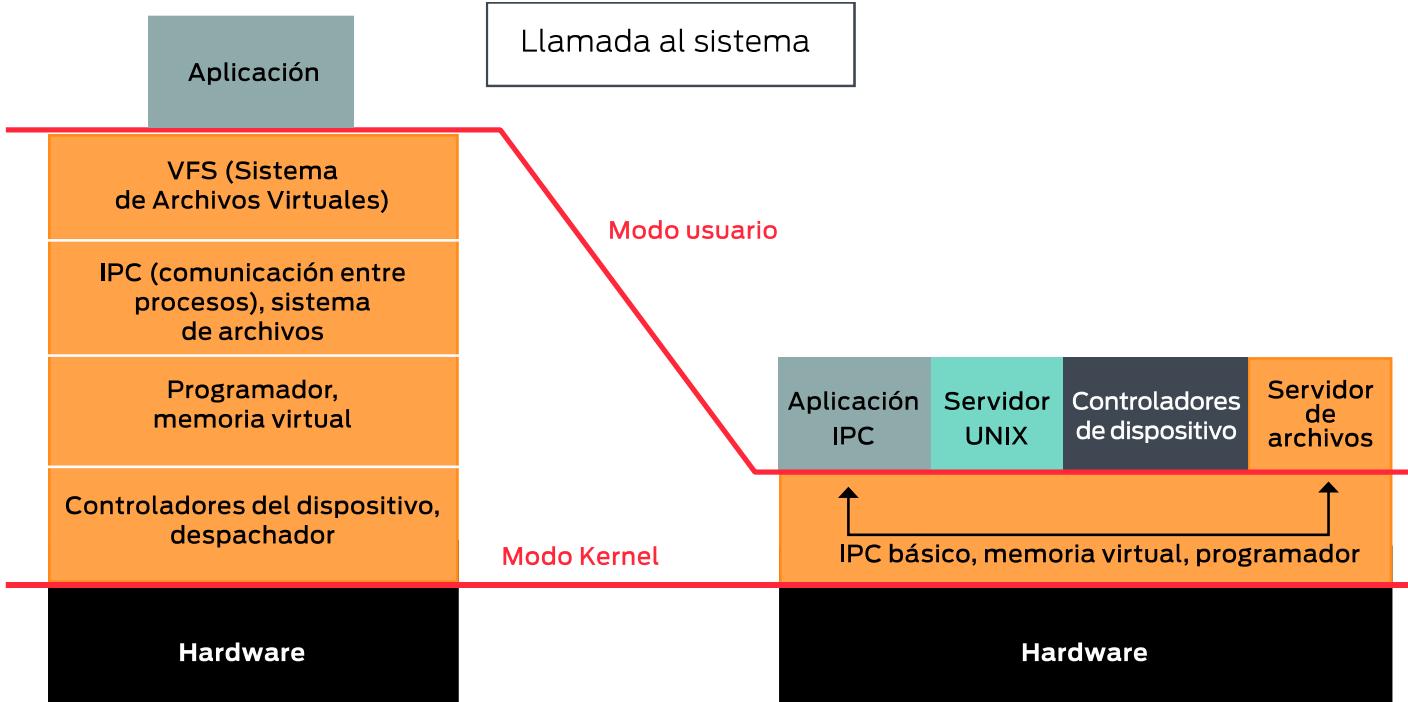
Sin embargo, lo anterior puede causar ciertos problemas, ya que mientras más procesos se ejecuten en modo supervisor, el sistema se vuelve más vulnerable a dejar de funcionar si uno no se comporta correctamente.



KERNEL

COMPONENTES PRINCIPALES

Sistema Operativo Monolítico basado en Kernel



¿Dónde se encuentran los archivos del Kernel de Linux?

Por lo regular, el archivo del *Kernel* se almacena en la carpeta **/boot** y se llama **vmlinuz-version**.

Cuando se desarrolló la memoria virtual para capacidades multitarea más sencillas, se colocó **vm** al principio del archivo. Esto para mostrar que el *Kernel* admite **memoria virtual**.





Durante un tiempo, el *Kernel* de Linux se llamó **vmlinu**x, pero creció demasiado para caber en la memoria de arranque disponible. Por ello, la imagen del *Kernel* se comprimió y la **x** final se cambió a **az**, para mostrar que estaba comprimida con compresión **zlib**. No obstante, esta misma compresión no siempre se usa; a menudo se reemplaza con **LZMA** o **BZIP2**; y algunos núcleos simplemente se llaman **zImage**.

La numeración de la versión tendrá el formato “ABCD”, donde “AB” probablemente será 2.6; “C” será su versión; y “D” indicará sus parches o correcciones.

En la carpeta **/boot** existen otros **archivos muy importantes**:

- **initrd.img-version**. Se usa como un pequeño disco RAM que extrae y ejecuta el archivo del *Kernel* real
- **system.map-version**. Se usa para administrar la memoria antes de que el núcleo se cargue por completo.
- **config-version**. Le dice al núcleo qué opciones y módulos debe cargar en la imagen del núcleo cuando se está compilando.

 config-2.6.35-22-generic	125.6 KB
 initrd.img-2.6.35-22-generic	10.3 MB
 memtest86+.bin	161.2 KB
 memtest86+_multiboot.bin	163.3 KB
 System.map-2.6.35-22-generic	1.7 MB
 vmcoreinfo-2.6.35-22-generic	1.2 KB
 vmlinuz-2.6.35-22-generic	4.1 MB



Arquitectura del Kernel de Linux

Antes que nada, cabe mencionar que, dado que el *Kernel* de Linux es monolítico, tiene una huella más grande y mayor complejidad sobre los otros tipos.

Esta fue una característica de diseño que estuvo bastante debatida en los primeros días de Linux, y aún tiene algunos de los mismos defectos de diseño que los núcleos monolíticos suelen tener.

Una mejora que los desarrolladores del *Kernel* de Linux hicieron para sortear estos defectos fue crear módulos que pudieran cargarse y descargarse durante el tiempo de ejecución. Esto significa ir más allá de simplemente agregar funcionalidad de hardware al *Kernel*, al incluir módulos que ejecutan procesos de servidor (como virtualización de bajo nivel), pero también puede permitir que se reemplace todo el *Kernel* sin necesidad de reiniciar tu computadora en algunos casos.

Imagina si pudieras actualizar a un paquete de servicio de Windows sin necesidad de reiniciar...

Módulos del núcleo

¿Qué pasaría si Windows tuviera todos los controladores disponibles ya instalados y solo tuviera que activar los controladores que necesita? Eso es esencialmente lo que hacen los módulos del *Kernel* para Linux. También conocidos como Módulos del Kernel Cargable (LKM), son esenciales para mantener el funcionamiento del *Kernel* con todo su hardware, sin consumir toda la memoria disponible.

```
ndiswrapper .ko
omnibook. ko
av5100 . ko
pbe5 . ko
r8192se_pci .ko
rothgar@ubuntu-vm:~$ ls -R /lib/modules/2.6.35-22-generic/kernel/ | grep -c .ko
3074
```

Un **módulo** generalmente agrega funcionalidad al *Kernel* base, para dispositivos, sistemas de archivos y llamadas al sistema. Los LKM tienen la extensión de archivo **.ko** y normalmente se almacenan en el directorio **/lib/modules**.

Debido a su naturaleza modular, se puede personalizar fácilmente, configurando los módulos para que se carguen o no, durante el inicio. Esto se realiza con el comando **menuconfig**, o editando el archivo **/boot/config**; o bien, puedes cargar y descargar módulos sobre la marcha con **modprobe**.





Los **módulos de código cerrado** y de terceros están disponibles en algunas distribuciones. Además, es posible que no se instalen de forma predeterminada, ya que el código fuente de los módulos no está disponible.

El desarrollador del software no proporciona el código fuente, sino que crea sus propios módulos y compila los archivos **.ko** necesarios para su distribución.

No obstante, un *Kernel* no es mágico, pero es completamente esencial para que cualquier computadora funcione correctamente. El *Kernel* de Linux es diferente al de otros SO, porque incluye controladores y hace que muchos procesos sean compatibles, es decir, que estén listos para usar.



LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 1

- Upsana. (2021). *Why do you need the different Linux Shells?* Edureka.
- Ramey, C. (2016). *The Architecture of Open Source Applications: The Bourne-Again Shell.* The Architecture of Open Source Applications.

LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 2

- Lichtmaier, N. (2015). *Arquitectura de Linux/UNIX - El kernel*. Kernel.
- R.H.E. (2017). *¿Qué es el Kernel de Linux?* Red Hat.



Te invitamos a realizar la siguiente actividad:

Presiona el botón para descargar la actividad:

Presiona el botón para entregar la actividad:



ACTIVIDAD INTEGRADORA #1



CONCLUSIÓN



Tanto el *Shell* como el *Kernel* en un sistema informático se utilizan para establecer la comunicación entre el hardware y el software, por lo que tienen una función fundamental en un sistema operativo. *Shell*, por una parte, es básicamente una interfaz presente entre el *Kernel* y el usuario. Además, permite a todos sus usuarios establecer comunicación con el *Kernel*. Mientras tanto, el *Kernel* es el núcleo mismo de un sistema operativo típico. Funciona para controlar todas las tareas que vienen con un sistema.



¡FELICIDADES!

Acabas de concluir la **segunda unidad** de tu curso *Sistemas Operativos I*. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

UNIDAD 3

GESTIÓN DE PROCESOS Y MEMORIA





3.1

TEMARIO

PROCESOS

3.2

MEMORIA



INTRODUCCIÓN



En esta unidad aprenderás los conceptos relacionados con la gestión de procesos y de memoria en Linux. En este sentido, conocerás más acerca de los procesos en primer y segundo plano. Además, realizarás un análisis más detallado de *Daemons*. Aunado a ello, conocerás el funcionamiento de las memorias física y virtual de Linux y sus respectivos comandos.



COMPETENCIAS A DESARROLLAR

- El alumno aprenderá el funcionamiento de procesos principales y secundarios en Linux, así como sus diferencias.

- El alumno comprenderá el funcionamiento de las memorias física y virtual de Linux, así como sus principales diferencias.



PROCESOS

Gestión de procesos Linux

Como hemos visto, Linux es un **sistema multitarea**, lo que significa que puede ejecutar múltiples procesos al mismo tiempo. Un **proceso**, por su parte, es un programa, una tarea o un comando en ejecución.

Existen 2 tipos de procesos en Linux:

- Procesos en primer plano
- Procesos en segundo plano





PROCESOS

Procesos en Primer Plano:

- También se conocen como procesos interactivos.
- Son aquellos que necesitan ser ejecutados o iniciados por el usuario o el programador. Es decir, **los servicios del sistema no pueden inicializarlos**.
- Toman la entrada del usuario para después devolver la salida. De esta manera, mientras estos se ejecutan no podemos iniciar directamente un nuevo proceso desde la misma terminal.

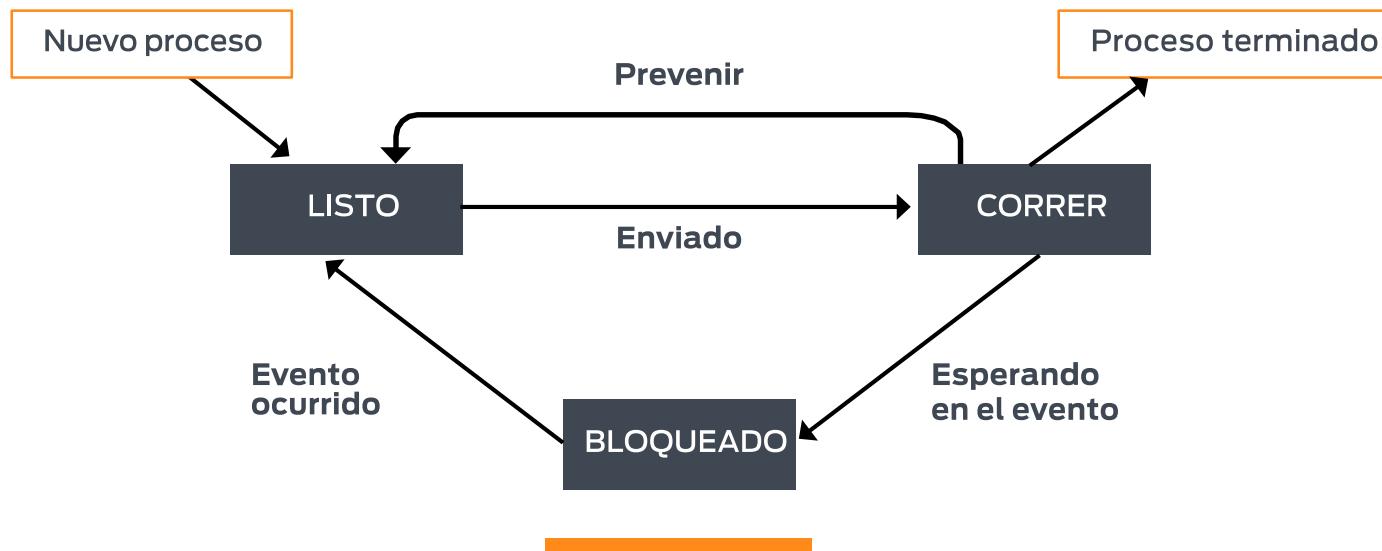
Procesos en Segundo Plano:

- También se conocen como procesos no interactivos.
- Son aquellos que **se ejecutan por el propio sistema o los usuarios.**
- Tienen un PID o Proceso Único si se les asigna. Además, podemos iniciar otros procesos dentro de la misma terminal desde la que se inician.



Daemons

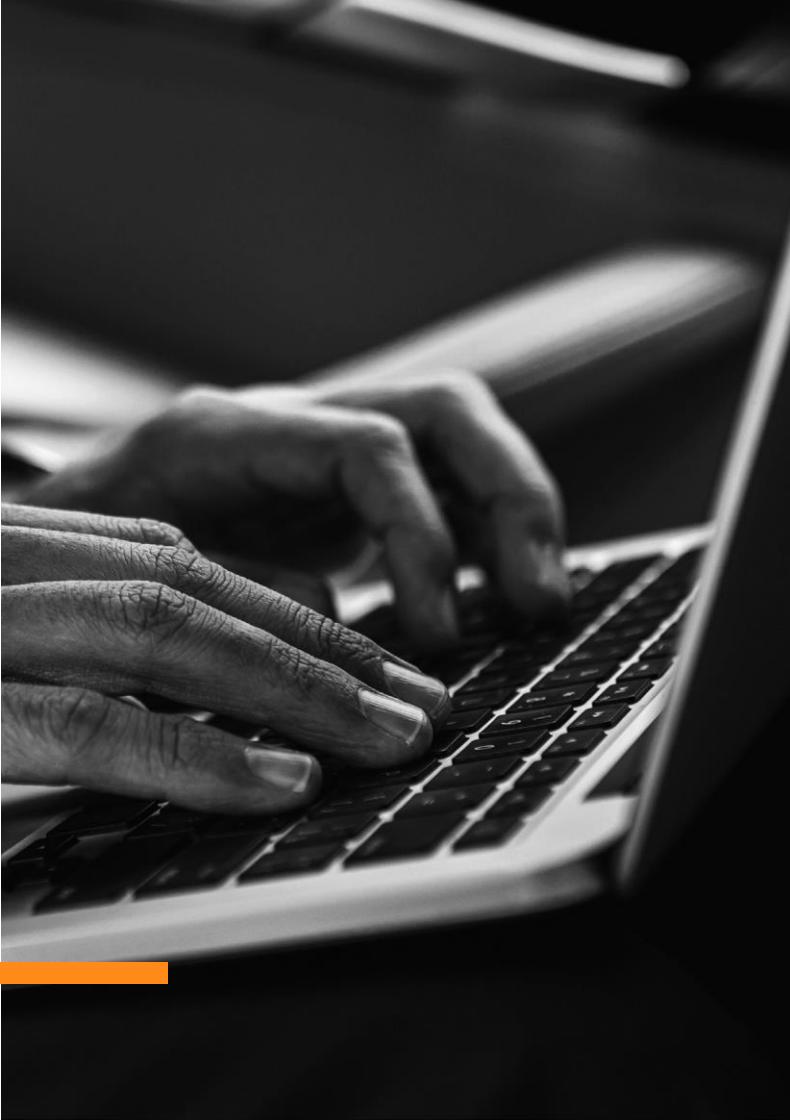
Como recordaremos, son tipos especiales de procesos en segundo plano. Estos se inician al mismo tiempo que el sistema, y lo siguen haciendo indefinidamente como **servicio**. Además, se ejecutan como servicios de forma espontánea. No obstante, los *Daemons* pueden ser controlados por un usuario a través del proceso **init**.



Estados de Procesos de Linux:

Durante la ejecución, un proceso cambia de un estado a otro, dependiendo de su entorno o circunstancias. A continuación, te presentamos los **estados posibles de un proceso en Linux**:

- En ejecución
- En espera
- Detenido
- Zombie





PROCESOS

- **En ejecución.** Es decir, se está ejecutando el proceso actual en el sistema; o bien, está listo para serlo. Está a la espera de ser asignado a una de las CPU.
- **En espera.** Aquí, un proceso está esperando un evento o recurso del sistema. A su vez, el *Kernel* diferencia entre **2 tipos de procesos de espera:**
 - Procesos de espera interrumpibles
 - Procesos de espera no interrumpibles



- **Detenido.** Esto quiere decir, en general, que el proceso se ha detenido al recibir una señal. Un ejemplo de esto es un proceso que se está depurando.
- **Zombie.** Se dice cuando un proceso está *muerto*; es decir, se ha detenido, pero todavía tiene una entrada en la tabla de procesos.





Creación de un Proceso en Linux

Se crea un nuevo proceso cuando uno ya existente realiza una copia exacta de sí mismo en la memoria. En este sentido, el **proceso hijo** tendrá el mismo entorno que el **proceso padre**. La única diferencia será su número de identificación.



Existen 2 formas para crear un nuevo proceso en Linux:

- **Función System().** Aunque este método es simple, puede llegar a ser ineficiente y con riesgos de seguridad significativos.
- **Funciones fork() y exec().** Esta técnica ofrece mayor flexibilidad, velocidad y seguridad.



Proceso de Inicio:

- Es el primer programa que se ejecuta cuando se inicia el sistema Linux.
- Gestiona todos los demás procesos del sistema y es iniciado por el propio *Kernel*, por lo que no tiene un *proceso padre*.
- Tiene un ID de proceso de “1”, por lo que es un *padre adoptivo* para todos los *procesos huérfanos*.
- Puedes usar el **comando pidof** para encontrar la ID de un proceso.

```
[root@tecmint ~] # pidof systemd  
1  
[root@tecmint ~] # pidof top  
2160  
[root@tecmint ~] # pidof httpd  
2103 2102 2101 2100 2099 1076  
[root@tecmint ~] #
```

Buscar el ID del proceso de Linux

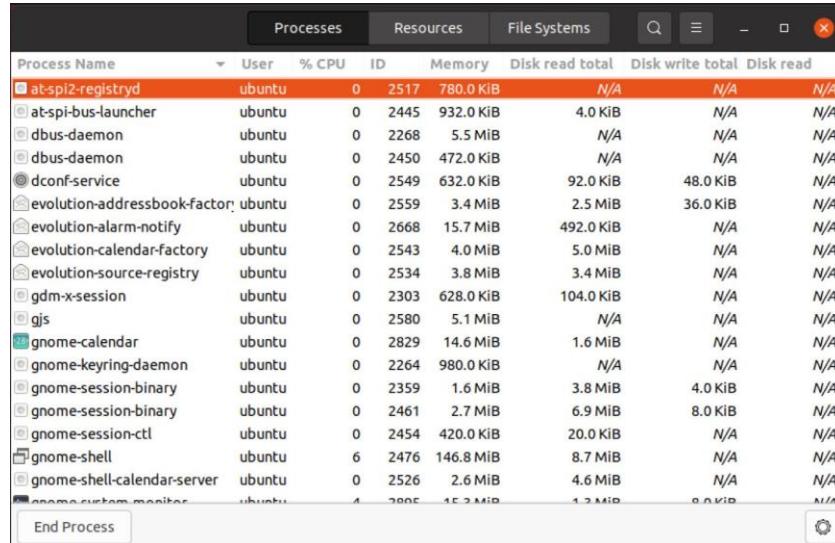
```
[root@tecmint ~] # echo $$  
2109  
[root@tecmint ~] # echo $PPID  
2106  
[root@tecmint ~] #
```

Encontrar el ID del proceso principal de Linux

Gestión de procesos con System Monitor

System Monitor ofrece una manera gráfica para mostrar y administrar los procesos. Para inicializarlo vamos a **Applications > System Monitor**.

Posteriormente, solo tenemos que cambiar a la pestaña *Process* para observar todos los procesos en ejecución.



Process Name	User	% CPU	ID	Memory	Disk read total	Disk write total	Disk read
at-spi2-registryd	ubuntu	0	2517	780.0 KIB	N/A	N/A	N/A
at-spi-bus-launcher	ubuntu	0	2445	932.0 KIB	4.0 KiB	N/A	N/A
dbus-daemon	ubuntu	0	2268	5.5 MiB	N/A	N/A	N/A
dbus-daemon	ubuntu	0	2450	472.0 KIB	N/A	N/A	N/A
dconf-service	ubuntu	0	2549	632.0 KIB	92.0 KiB	48.0 KiB	N/A
evolution-addressbook-factor	ubuntu	0	2559	3.4 MiB	2.5 MiB	36.0 KiB	N/A
evolution-alarm-notify	ubuntu	0	2668	15.7 MiB	492.0 KiB	N/A	N/A
evolution-calendar-factory	ubuntu	0	2543	4.0 MiB	5.0 MiB	N/A	N/A
evolution-source-registry	ubuntu	0	2534	3.8 MiB	3.4 MiB	N/A	N/A
gdm-x-session	ubuntu	0	2303	628.0 KIB	104.0 KiB	N/A	N/A
gjs	ubuntu	0	2580	5.1 MiB	N/A	N/A	N/A
gnome-calendar	ubuntu	0	2829	14.6 MiB	1.6 MiB	N/A	N/A
gnome-keyring-daemon	ubuntu	0	2264	980.0 KIB	N/A	N/A	N/A
gnome-session-binary	ubuntu	0	2359	1.6 MiB	3.8 MiB	4.0 KiB	N/A
gnome-session-binary	ubuntu	0	2461	2.7 MiB	6.9 MiB	8.0 KiB	N/A
gnome-session-ctl	ubuntu	0	2454	420.0 KIB	20.0 KiB	N/A	N/A
gnome-shell	ubuntu	6	2476	146.8 MiB	8.7 MiB	N/A	N/A
gnome-shell-calendar-server	ubuntu	0	2526	2.6 MiB	4.6 MiB	N/A	N/A
gnome-system-monitor	ubuntu	1	2805	15.2 MiB	1.2 MiB	0.0 KiB	N/A

PROCESOS

Process Name	User	% CPU	ID	Memory	Disk read total	Disk write total	Disk read
at-spi2-registryd	ubuntu	0	2517	780.0 KiB	N/A	N/A	N/A
at-spi-bus-launcher	Properties	Alt+Return	445	932.0 KiB	4.0 KiB	N/A	N/A
dbus-daemon	Memory Maps	Ctrl+M	268	5.5 MiB	N/A	N/A	N/A
dbus-daemon	Open Files	Ctrl+O	450	472.0 KiB	N/A	N/A	N/A
dconf-service	Change Priority	▶	549	632.0 KiB	92.0 KiB	48.0 KiB	N/A
evolution-addressbook	Stop	Ctrl+S	559	3.4 MiB	2.5 MiB	36.0 KiB	N/A
evolution-alarmclock	Continue	Ctrl+C	568	15.7 MiB	492.0 KiB	N/A	N/A
evolution-calendar	End	Ctrl+E	543	4.0 MiB	5.0 MiB	N/A	N/A
evolution-sourcebook	Kill	Ctrl+K	534	3.8 MiB	3.4 MiB	N/A	N/A
gdm-x-session			303	628.0 KiB	104.0 KiB	N/A	N/A

System Monitor muestra los procesos en orden alfabético para el usuario actual. Además, se puede reordenar la lista de procesos dando clic en cualquier encabezado de columna. Aunado a ello, podemos hacer clic en los encabezados *% CPU* y *Memoria* para ver los procesos que consumen más potencia de procesamiento y memoria.



PROCESOS

Además de lo anterior, *System Monitor* permite **administrar procesos de varias maneras**. Esto quiere decir que los procesos se pueden detener, suspender, continuar, finalizar y cambiar de prioridad simplemente dando clic derecho en un nombre de proceso.

Así, *System Monitor* representa la mejor herramienta para los usuarios novatos que todavía están aprendiendo cómo trabajar con utilidades de línea de comandos.



VIDEO

Te invitamos a ver el siguiente video:



MEMORIA

Antes que cualquier cosa, es importante conocer que existen **2 tipos de memorias en Linux:**

- Memoria Física
- Memoria Virtual



Memoria Física:

- Se trata de la memoria real presente en la máquina.
- También se denomina memoria principal o mapa de memoria.
- Es la memoria dinámica de acceso aleatorio (DRAM).
- Solo el *Kernel* tiene acceso directo a la memoria física.

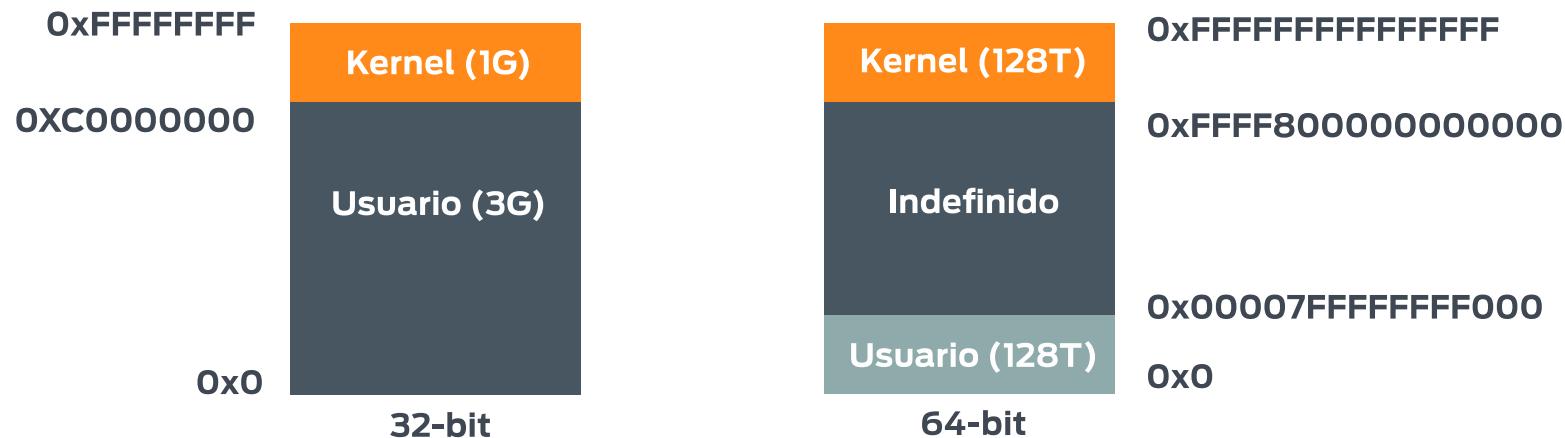




MEMORIA

El *Kernel* de Linux proporciona a cada proceso un espacio de direcciones virtual independiente; además, este es contiguo. De esta forma, el proceso puede acceder fácilmente a la memoria; más precisamente, a la memoria virtual.

El **interior del espacio de direcciones virtuales** se divide en 2 partes: espacio del **Kernel** y **espacio del usuario**. Los procesadores con diferente longitud de palabra (longitud máxima de datos que puede procesar una sola instrucción de CPU) tienen diferentes rangos de espacio de direcciones. Por ejemplo, para sistemas de 32 y 64 bits, el siguiente diagrama muestra su espacio de memoria virtual:



Espacio del núcleo del sistema, espacio del Kernel y el espacio del usuario del sistema



MEMORIA

Memoria Virtual:

- Suele ser mayor que la memoria física.
- El *Kernel* de Linux utiliza la memoria virtual para dejar que los programas reserven memoria.
- Mientras se ejecuta un programa, el procesador lee las instrucciones de la memoria virtual. Sin embargo, antes de ejecutarlas, convierte las direcciones virtuales en direcciones físicas.

Gestión de memoria

Se encarga de administrar la memoria en el sistema.
Esto **incluye**:

- La implementación de la memoria virtual y la paginación por demanda.
- La asignación de memoria tanto para las estructuras internas del *Kernel* como para los programas de espacio del usuario.
- La asignación de archivos en el espacio de direcciones de los procesos. Entre otras acciones.



Descripción de la gestión de memoria de Linux:

- 1 ► La parte central de la computadora es la CPU; y la RAM es el portal frontal de la CPU.
- 2 ► Todo lo que va a la CPU pasará por la RAM. Por ejemplo, si tenemos un proceso que se está cargando, este primero se cargará en la RAM y la CPU obtendrá los datos del proceso de la RAM.
- 3 ► No obstante, para hacerlo más rápido, la CPU tiene caché de nivel uno, dos y tres. Eso es como RAM, pero en la CPU.

- 4 ► Hay cantidades muy pequeñas de caché en la CPU, ya que es muy caro y tampoco es muy útil para todas las instrucciones.
- 5 ► Por lo tanto, la información del proceso se copiará de la RAM a la CPU, y esta última construirá su caché.
- 6 ► Aquí el caché también juega un papel importante en la gestión de la memoria en Linux. No obstante, para hacerlo más rápido, la CPU tiene caché de nivel uno, dos y tres. Eso es como RAM, pero en la CPU.

- 7 ► Si un usuario solicita información desde el disco duro, esta se copia a la RAM y el usuario recibirá el servicio desde esta.
- 8 ► Y aunque la información se copia del disco duro, se coloca en lo que llamamos **caché de página**.
- 9 ► Por lo tanto, el caché de la página almacena los datos solicitados recientemente para acceder a ellos más rápidamente cuando se requiera.

10

- ▶ Así, si el usuario comienza a modificar los datos, también irá a la RAM; y desde esta se copiará al disco duro. No obstante, eso solo sucederá si los datos han estado en la RAM el tiempo suficiente.

11

- ▶ Los datos no se escribirán inmediatamente desde la RAM al disco duro. Sin embargo, para optimizar la escritura en el disco duro, Linux funciona con el concepto de caché sucio.

12

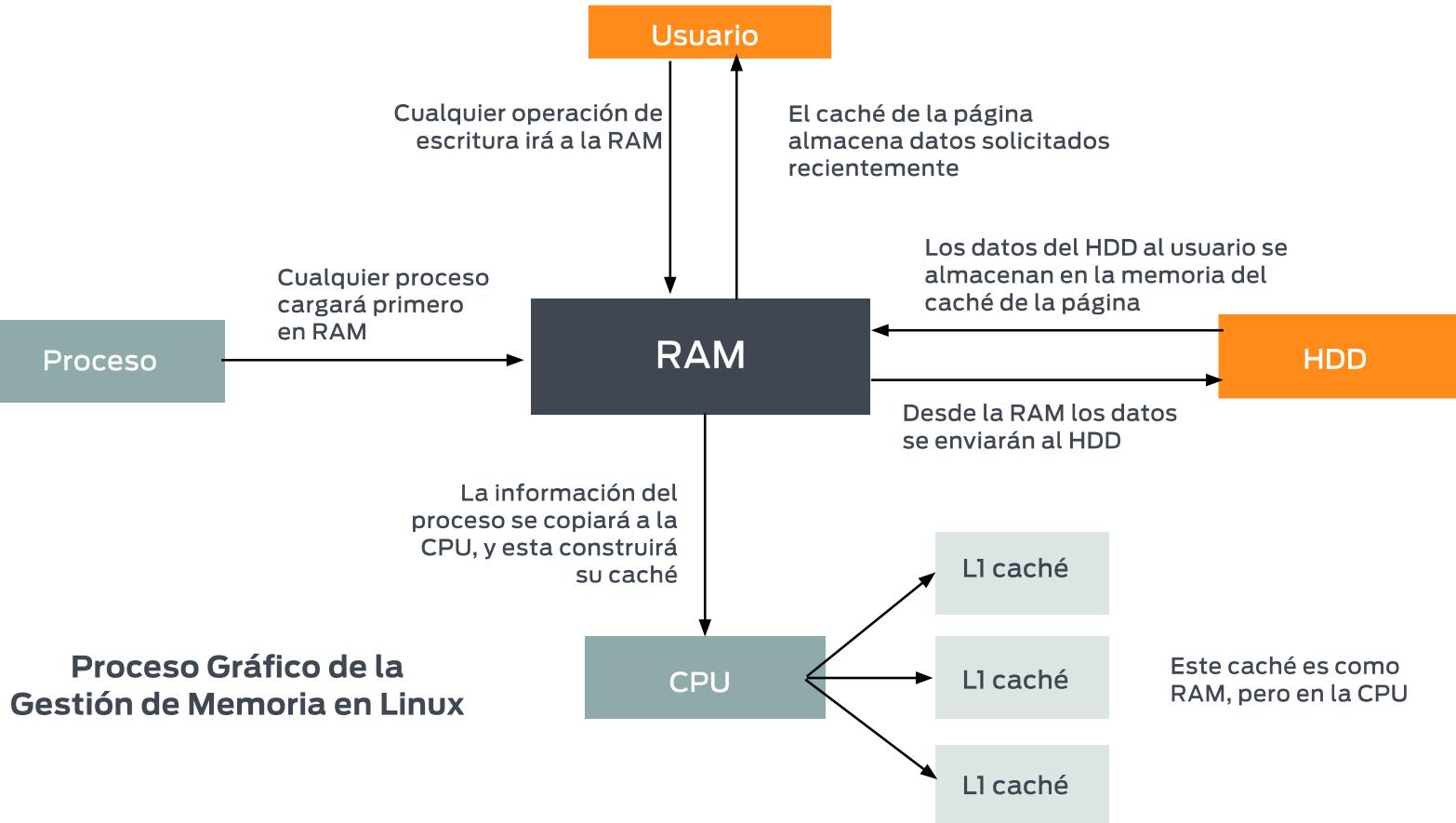
- ▶ Se intentan almacenar en el búfer los datos para crear una solicitud de escritura eficiente.

13

- ▶ Luego, todo lo que sucede en tu computadora pasa por la RAM.

14

- ▶ Por lo tanto, **usar RAM en una computadora con Linux es esencial para el buen funcionamiento del sistema operativo.**



Conozcamos algunos de los **comandos para administrar la memoria en Linux**:

1 ► **/proc/meminfo**. Este archivo contiene toda la información relacionada con la memoria. Para conocerlo se use el comando **cat**:

```
$ cat /proc/meminfo
```

```
LinuxForDevices> cat /proc/meminfo
MemTotal:      2035428 kB
MemFree:       205284 kB
MemAvailable:  1585480 kB
Buffers:        191332 kB
Cached:         1271296 kB
SwapCached:    136 kB
Active:         835336 kB
Inactive:      744580 kB
Active(anon):   61996 kB
Inactive(anon): 63612 kB
Active(file):   773340 kB
Inactive(file): 680968 kB
Unevictable:   18560 kB
Mlocked:        18560 kB
SwapTotal:     524284 kB
SwapFree:      513736 kB
Dirty:          1064 kB
Writeback:      0 kB
AnonPages:     135768 kB
Mapped:         121864 kB
Shmem:          928 kB
KReclaimable:  113628 kB
Slab:           195740 kB
SReclaimable:  113628 kB
SUnreclaim:    82112 kB
KernelStack:   2940 kB
PageTables:    3140 kB
NFS_Unstable:  0 kB
Bounce:         0 kB
WritebackTmp:   0 kB
Commitlimit:   1541996 kB
Committed_AS:  950644 kB
VmallocTotal:  34359738367 kB
VmallocUsed:   22704 kB
VmallocChunk:  0 kB
Percpu:         864 kB
HardwareCorrupted: 0 kB
AnonHugePages:  0 kB
ShmemHugePages: 0 kB
ShmemPmdMapped: 0 kB
FileHugePages:  0 kB
FilePmdMapped: 0 kB
CmaTotal:      0 kB
CmaFree:        0 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize:   2048 kB
Hugetlb:        0 kB
DirectMap4k:    155504 kB
DirectMap2M:    1941584 kB
DirectMap1G:    0 kB
```

Para **obtener la memoria física** del uso del archivo **/proc/meminfo** se utiliza:

```
1      $ grep MemTotal /proc/meminfo
```

```
[LinuxForDevices]> grep MemTotal /proc/meminfo  
MemTotal:      2035428   kB
```

Para **obtener la memoria virtual** del archivo **/proc/meminfo** se utiliza:

```
1      $ grep VmallocTotal /proc/meminfo
```

```
[LinuxForDevices]> grep VmallocTotal /proc/meminfo  
VmallocTotal:      34359738367   kB
```



2 ► **top.** Permite monitorear los procesos y el uso de recursos del sistema en Linux. Además, ofrece una vista dinámica en tiempo real del sistema. Cuando ejecutes este comando notarás que los valores en la salida siguen cambiando. Esto sucede debido a que se muestran los valores en tiempo real:

```
1      $ top
```

MEMORIA

Así, en la parte superior se muestran las estadísticas de uso actuales de los recursos del sistema. Por otro lado, en la parte inferior está la información sobre los procesos en ejecución.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	169312	13144	8064	S	0.0	0.6	2:46.93	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.55	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-kblockd
9	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
10	root	20	0	0	0	0	S	0.0	0.0	1:24.80	ksoftirqd/0
11	root	20	0	0	0	0	I	0.0	0.0	3:14.60	rcu_sched
12	root	rt	0	0	0	0	S	0.0	0.0	0:48.56	migration/0
13	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
16	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
17	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_kthre
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kaudit
19	root	20	0	0	0	0	S	0.0	0.0	0:03.17	khungtaskd
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	oom_reaper
21	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	writeback
22	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kcompactd0
23	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
24	root	39	19	0	0	0	S	0.0	0.0	0:11.51	khugepaged
116	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kintegrityd
117	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kblockd
118	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	blkcg_punt_bio
119	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	tpm_dev_wq
120	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ata_sff
121	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	md
122	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	edac-poller
123	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	devfreq_wq
124	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	watchdogd
127	root	20	0	0	0	0	S	0.0	0.0	0:29.14	kswapd0
128	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cryptfs-kthrea
131	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kthrotld
132	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	acpi_thermal_pm
133	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	vfio-irqfd-clea
134	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ipv6_addrconf
145	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kstrt
149	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/u3:0
165	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	charger_manager
284	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_0
285	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	scsi_tmf_0
286	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_1
287	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	scsi_tmf_1
289	root	0	-20	0	0	0	I	0.0	0.0	0:51.34	kworker/0:1H-kblockd
210	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_2
211	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	scsi_tmf_2
212	root	20	0	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_3

3 ► **free.** Este comando muestra tanto la cantidad de memoria libre como la memoria utilizada que hay en el sistema. Además, también te informa sobre la cantidad total de memoria física y de intercambio en tu sistema:

```
1      $ free
```

```
[LinuxForDevices]# free
              total        used        free      shared  buff/cache   available
Mem:       2035428      253696      205244          928      1576488      1585672
Swap:      524284           0      513736
```



► **vmstat.** Es una herramienta de monitoreo de rendimiento en Linux. Brinda información útil sobre procesos, memoria, E/S de bloque, paginación, disco y programación de CPU. Además, informa sobre estadísticas de memoria virtual de tu sistema:

```
1      $ vmstat
```

```
[LinuxForDevices]# vmstat
procs      --memory--  --swap--  --io--  -system--  --cpu--
 r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
 0 0 10548 204992 191340 1385176  0   0   0    9    4    4   0   0 100   0   0
```

LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 1

- GeeksforGeeks. (2020). Process Management in Linux.
- A.K. (2017). *All You Need To Know About Processes in Linux [Comprehensive Guide]*. TechMint #1 Linux Blog.



LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 2

- T. (2022). Linux — *How Does Memory Management Work?* - Geek Culture. Medium.
- U. (2020). Memory Management — *The Linux Kernel documentation*. Docs Kernel.





ACTIVIDAD INTEGRADORA #2

Te invitamos a realizar la siguiente actividad:

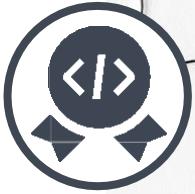
Presiona el botón para descargar la actividad:

Presiona el botón para entregar la actividad:



CONCLUSIÓN

La gestión de procesos y de memoria son conceptos fundamentales y necesarios al aprender a trabajar con Linux. En este sentido, conocer cómo administrar un proceso de Linux nos pone un paso más cerca de dominar este sistema tan importante. Cuestiones como cuáles son los procesos que consumen la mayor parte del uso de CPU y RAM, cómo administrarlos, aumentar la velocidad y el rendimiento del sistema brindarán un mejor entorno para ejecutar cualquier proceso que se desee de manera más eficiente.

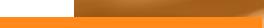


¡FELICIDADES!

Acabas de concluir la **tercera unidad** de tu curso *Sistemas Operativos I*. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

UNIDAD 4

GESTIÓN DE ENTRADAS, SALIDAS Y SISTEMA DE ARCHIVOS





4.1

TEMARIO

ENTRADAS
Y SALIDAS

4.2

SISTEMA
DE ARCHIVOS



INTRODUCCIÓN

En esta unidad aprenderás acerca de los dispositivos de entrada y salida de los equipos de cómputo en general. A su vez, conocerás cómo es que estos se pueden administrar en el sistema operativo de Linux.

Además, comprenderás acerca del sistema de gestión de archivos, sus tipos y cómo estos se pueden administrar, asemejándose a una ramificación de árbol.

COMPETENCIAS A DESARROLLAR



- El alumno comprenderá la administración de entradas y salidas en Linux.

- El alumno comprenderá la administración del gestor de archivos de Linux.

ENTRADAS Y SALIDAS

Para un usuario, el **Sistema de E/S** (entradas y salidas) en Linux se parece mucho al de cualquier sistema UNIX. En este sentido, en la medida de lo posible, **todos los controladores de dispositivos aparecen como archivos normales**.

Así, un usuario puede abrir un canal de acceso a un dispositivo de la misma manera que abre cualquier otro archivo: los dispositivos pueden aparecer como objetos dentro del sistema de archivos.





Los dispositivos de E/S también se conocen como **periféricos**. Lo anterior se debe a que estos no forman parte total del sistema informático en sí, sino que son elementos que se conectan a través de las ranuras de entrada o salida. Existen **3 tipos de dispositivos**:

- Entrada
- Salida
- Entradas/Salidas

- **Entrada.** Estos dispositivos envían información a un sistema para su posterior procesamiento. Después, un dispositivo de salida reproduce los resultados de dicho procesamiento. Además, los dispositivos de entrada, como su nombre lo dice, solo permiten la entrada de datos a una computadora. Ejemplos:

- Mouse
- Teclado
- Cámara web
- Micrófono, entre otros.

ENTRADAS Y SALIDAS



- **Salida.** Estos dispositivos pueden recibir datos de otro dispositivo y generar salida con dichos datos; sin embargo, no pueden enviar datos a otro dispositivo. Ejemplos:

- Impresora
- Monitor
- Proyector
- Altavoces, entre otros.



Proyector



Altavoces



Auriculares

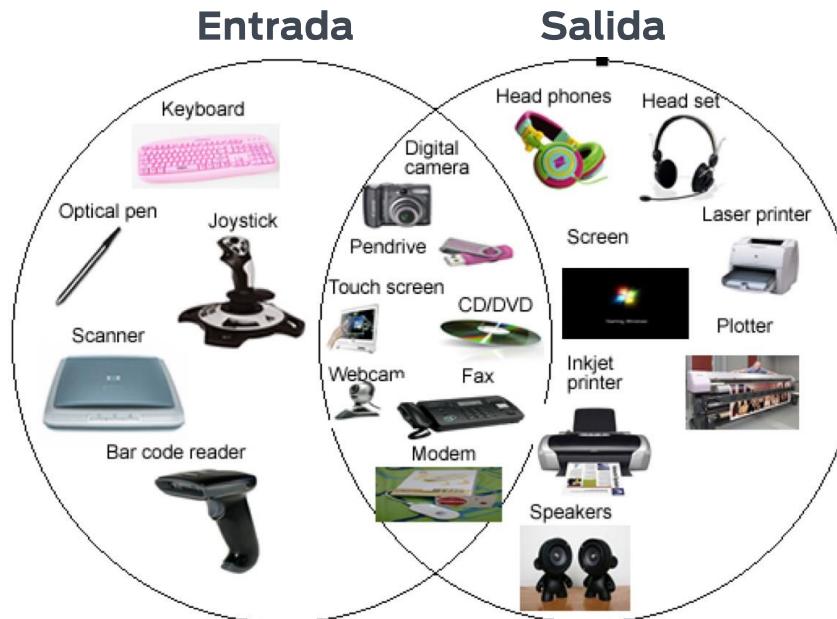


Impresora



Monitor

- **Entrada/Salida.** Estos dispositivos (E/S) tienen la capacidad de aceptar datos de entrada, salida u otros datos procesados. Además, permiten adquirir datos como entrada; o bien, enviar datos de computadora a medios de almacenamiento, como salida de almacenamiento. Este tipo de dispositivos se conocen como dispositivos IO.



Un **administrador del sistema** puede crear archivos especiales dentro de un sistema de archivos. Estos pueden contener referencias a un controlador de dispositivo específico. Así, cualquier usuario que abra este archivo podrá leer y escribir en dicho dispositivo.

Mediante el uso del **sistema normal de protección de archivos** el administrador puede establecer permisos de acceso para cada dispositivo. Linux divide todos los dispositivos en **3 clases**:

- Dispositivos de bloque,
- Dispositivos de caracteres y
- Dispositivos de red.

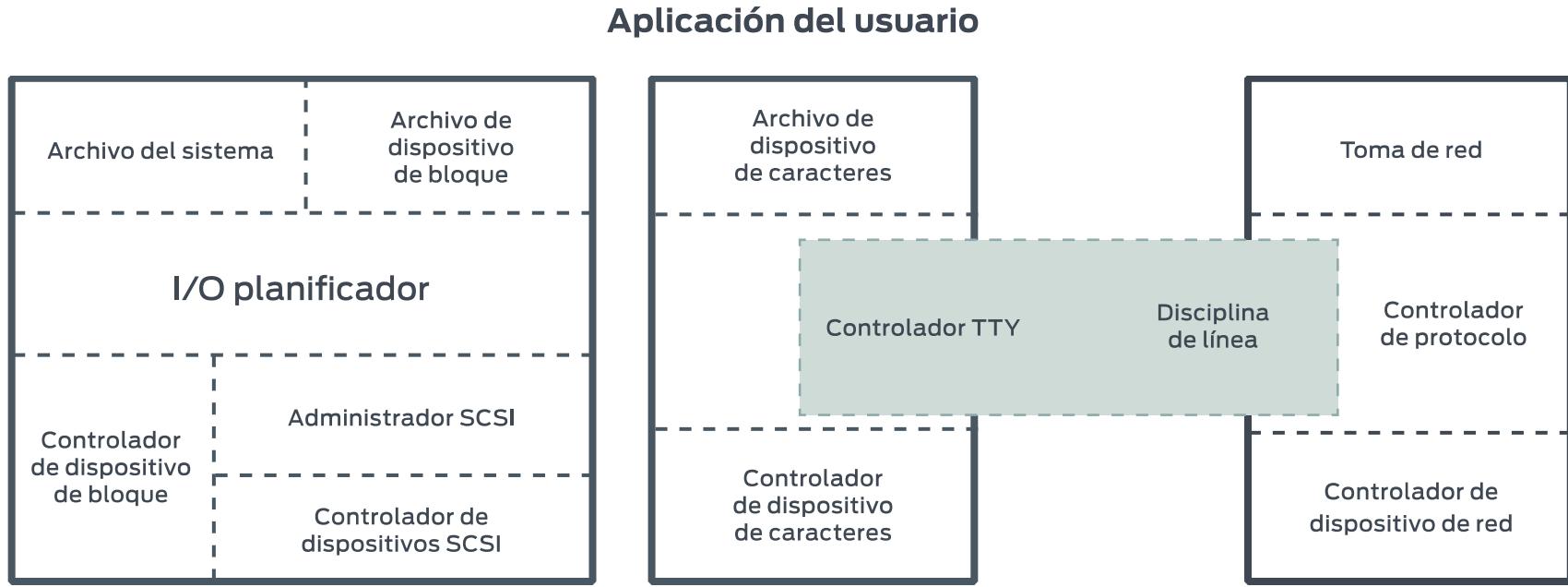


Figura: Estructura en bloque del controlador de dispositivos

- **Dispositivos de bloque.** Son aquellos dispositivos que permiten el acceso aleatorio a bloques de datos de tamaño fijo completamente independientes.

Los dispositivos de bloque se utilizan normalmente para almacenar sistemas de archivos. No obstante, también se permite el acceso directo a un dispositivo de bloque para que los programas puedan crear y reparar el sistema de archivos que contiene el dispositivo.

Así, las aplicaciones pueden acceder a los dispositivos de bloque directamente si lo desean.

Los dispositivos de caracteres, por su parte, incluyen la mayoría de los demás dispositivos, como ratones y teclados.

La diferencia fundamental entre los dispositivos de bloque y los de caracteres es el **acceso aleatorio**. Es decir, se puede acceder a los dispositivos de bloque de forma aleatoria, mientras que a los dispositivos de caracteres solo se accede en serie.



ENTRADAS Y SALIDAS

GESTIÓN DE ENTRADAS, SALIDAS
Y SISTEMA DE ARCHIVOS

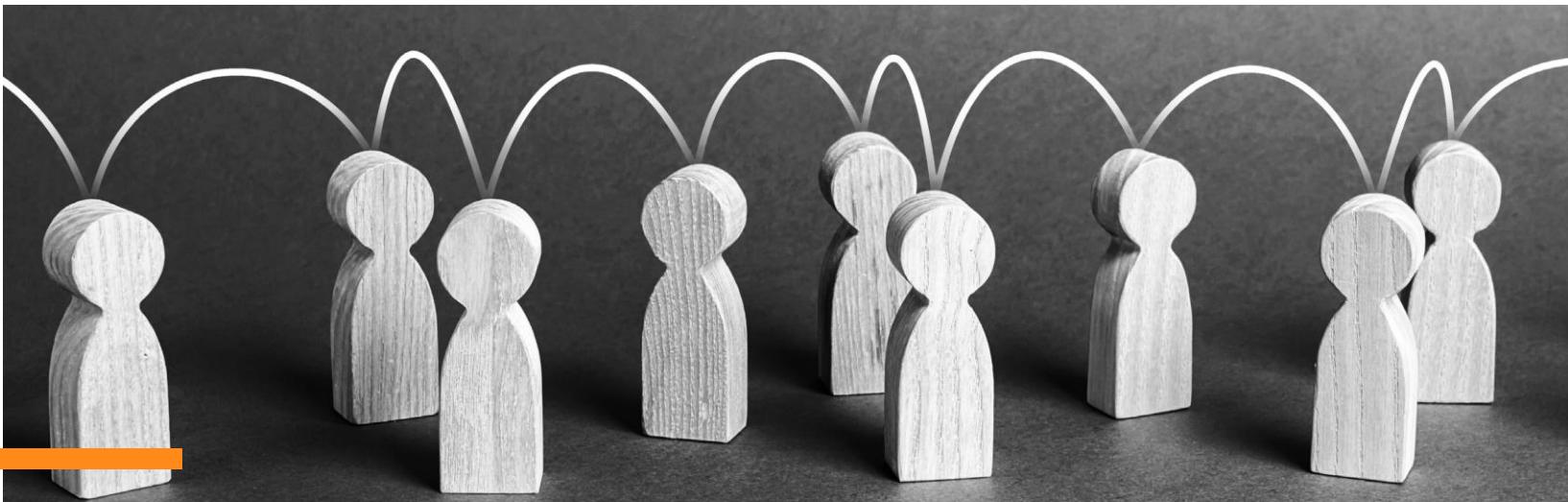


Los dispositivos de bloque proporcionan la interfaz principal para todos los dispositivos de disco en un sistema. El rendimiento es particularmente importante para los discos, y el sistema de dispositivos de bloques debe proporcionar funcionalidad para garantizar que el acceso a este sea lo más rápido posible. Esto se logra a través de la **programación de operaciones de E/S**.

En este contexto, un bloque representa la unidad con la que el núcleo realiza E/S. Así, cuando un bloque se lee en la memoria, se almacena en un búfer.

Cuando un controlador de dispositivo de bloque acepta una solicitud para su procesamiento, esta no se elimina de la lista, sino después de que se completa la E/S. En este momento, el controlador continúa con la siguiente solicitud de la lista, incluso si se han insertado nuevas solicitudes antes de la solicitud activa.

A medida que se realizan nuevas solicitudes de E/S, el administrador de solicitudes intenta fusionarlas en las listas por dispositivo.



- **Dispositivos de caracteres.** Puede ser casi cualquier controlador de dispositivo que no ofrezca acceso aleatorio a bloques fijos de datos. Así, todo controlador registrado en el *Kernel* de Linux también debe registrar un conjunto de funciones que implementen las operaciones de E/S de archivos. El *Kernel* casi no realiza preprocesamiento de una solicitud de lectura o escritura de un archivo en un dispositivo de caracteres, ya que solo pasa la solicitud al dispositivo en cuestión y deja que este se ocupe de ello.



ENTRADAS Y SALIDAS

La **principal excepción a la anterior regla** es el subconjunto especial de controladores de dispositivos de caracteres que implementan dispositivos terminales.

En este sentido, el *Kernel* mantiene una interfaz estándar para estos controladores a través de un conjunto de estructuras **`tty_struct`**. Cada una de las cuales proporciona almacenamiento en búfer, así como control en el flujo de datos desde el dispositivo terminal, y alimenta esos datos a una disciplina de línea.

Disciplina de línea. Es un intérprete de la información del dispositivo terminal. La más común es la **disciplina tt**, la cual vincula el flujo de datos de la terminal con los flujos de entrada y salida estándar de los procesos en ejecución de un usuario. Lo anterior permite que dichos procesos se comuniquen directamente con la terminal del usuario.



No obstante, este trabajo se complica debido a que varios de estos procesos pueden estar ejecutándose simultáneamente, por lo que la disciplina de línea tt es responsable de activar y desactivar la entrada y salida de la terminal de los diversos procesos adjuntos cuando el usuario suspende; o bien, de activar esos procesos.



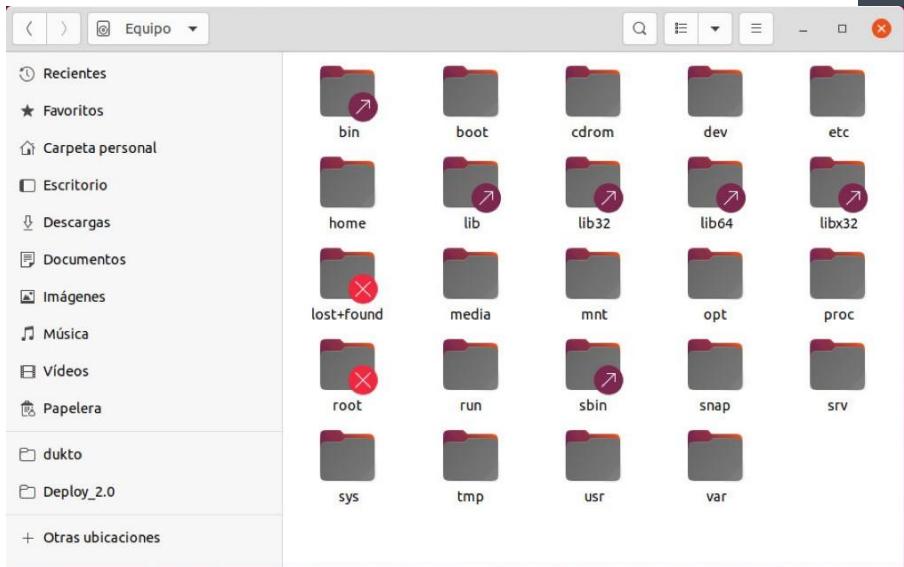
VIDEO

Te invitamos a ver el siguiente video:



SISTEMA DE ARCHIVOS

Los orígenes del sistema de archivos de Linux se encuentran en el origen general del *Kernel* y del sistema operativo Unix en general. A diferencia de Windows o MS-DOS, el sistema de archivos de cualquier sistema operativo derivado de Unix no está directamente relacionado con la estructura del hardware o, más comúnmente, con la cantidad de discos asociados con una computadora.





En Linux, todo el sistema de archivos comienza desde una **root (raíz)**, la cual se representa por un “**/**”. Todos los demás archivos están por abajo de esta raíz, en forma de árbol.

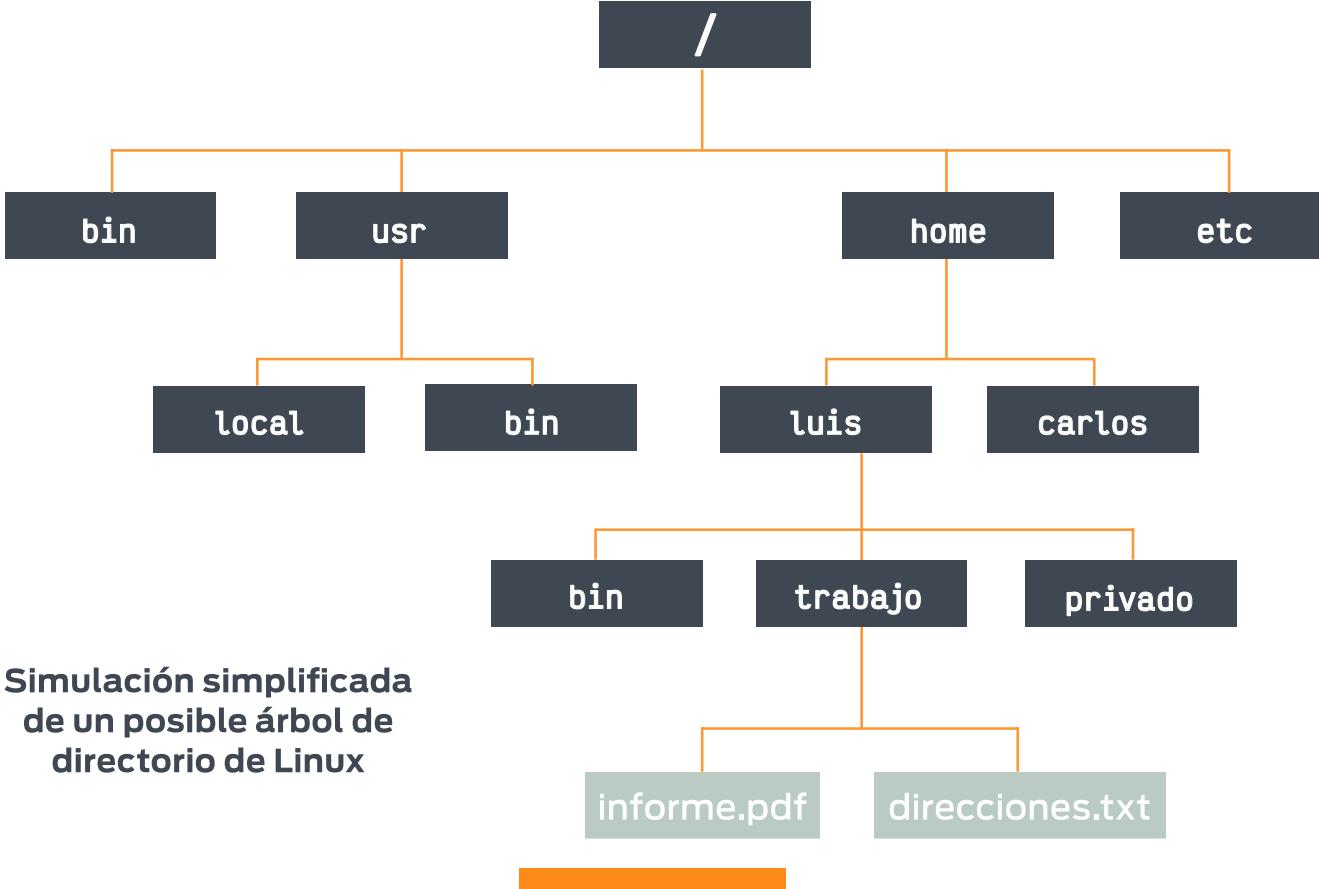
La raíz en Linux **no contiene ningún archivo**. Esta es una **regla general**.

En un sistema de este tipo de archivos, estos se organizan en **directorios** (son más un tipo especial de archivo que agrupa a otros). Dado que un directorio también puede contener otros directorios, la estructura del sistema de archivos se asemeja a un **árbol**.

Esto motivó a desarrollar lo que se conoce como **FHS (*Filesystem Hierarchy Standard*)**.



SISTEMA DE ARCHIVOS



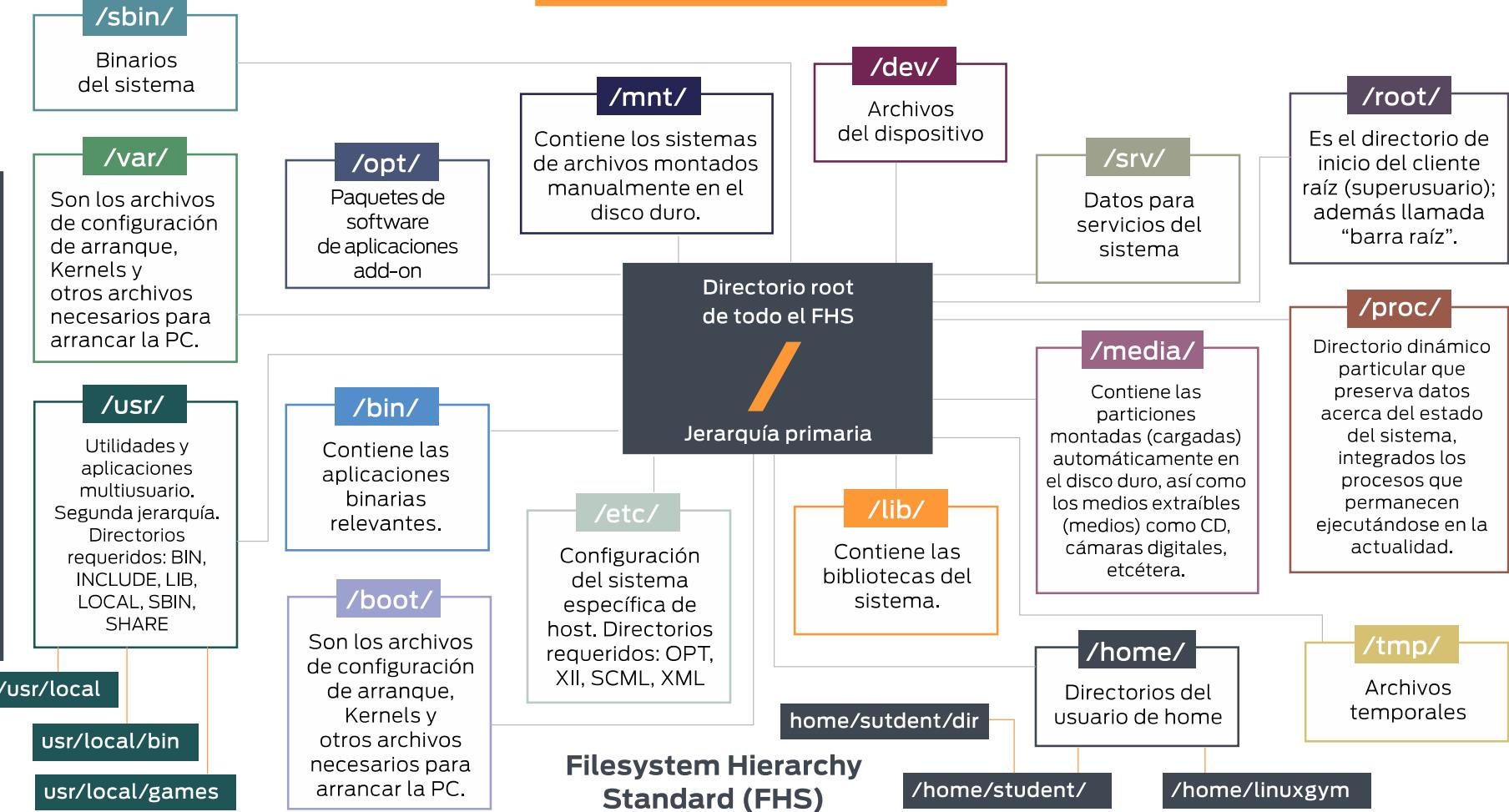


FHS (Filesystem Hierarchy Standard):

- Se define como el estándar que establece y proporciona el detalle de los nombres, contenidos, permisos y autorizaciones de archivos y directorios.
- Es el conjunto de reglas que determina una estructura de archivos y directorios.
- Es un documento guía, el cual puede ser consultado por los fabricantes, y aplicado a la hora de crear una nueva distribución.

SISTEMA DE ARCHIVOS

GESTIÓN DE ENTRADAS, SALIDAS
Y SISTEMA DE ARCHIVOS



La ventaja de integrar el estándar FHS en tu sistema Linux es que el entorno será mucho **más compatible** con otras distribuciones de Linux.

Además, este estándar permite cierta flexibilidad. Por lo tanto, existen ciertas libertades a la hora de aplicar las reglas. He ahí la razón de que se mantengan diferencias leves entre las distintas distribuciones.





SISTEMA DE ARCHIVOS

Principales objetivos del FSE:

- Exponer un sistema de archivos jerárquico de manera consistente y uniforme.
- Facilitar el desarrollo de software, ya que se pueden predecir e identificar fácilmente los archivos y directorios instalados.
- Brinda al usuario la posibilidad de predecir la ubicación de archivos y directorios en su computadora.

El **enfoque primordial del FHS** es la creación de sistemas operativos con estructuras sumamente compatibles. Esto ofrecerá una mejor experiencia para todos los usuarios. En consecuencia, estos podrán comprender el significado de cada elemento en el sistema, así como su ubicación.

Por otro lado, el propio FHS muestra los **tipos de archivos** que se pueden considerar en la estructura del sistema:

- Archivos compatibles y no compatibles
- Archivos estáticos y variables





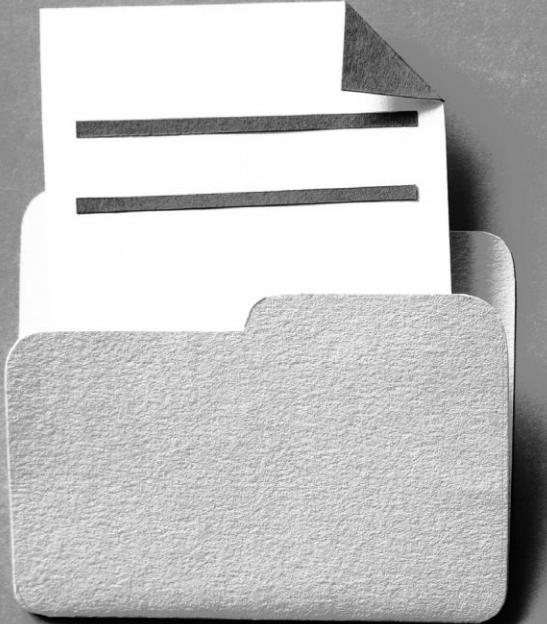
Archivos compatibles y no compatibles. Los primeros son aquellos archivos que pertenecen a una computadora; mientras que los archivos no compatibles son aquellos que se pueden compartir entre diferentes computadoras. Por ejemplo:

- **Archivos compatibles.** Los contenidos en `/var/www/html` (`DocumentRoot` predeterminado del servidor web Apache). Aquí es donde se almacena inicialmente el `index.html` de bienvenida.
- **Archivos no compatibles.** Como los contenidos en `/boot/grub/` (subdirectorio donde se encuentran los archivos del gestor de arranque GRUB).

Archivos estáticos y variables. Los primeros son aquellos que requieren la interacción del administrador del sistema para cambiar de estado. Las variables, por su parte, pueden cambiar sin tal interacción. Ejemplo:

Tenemos los archivos de registro del sistema o *logs* (de tipo **variable**). Estos se modifican constantemente sin la intervención del administrador, ya que son mensajes generados por el *Kernel* del sistema. Mientras que el resto de archivos donde se almacena información sensible, como cuentas de usuario, son de tipo **estático**.





Acceso a diferentes sistemas de archivos:

Tanto el hardware como el programa se almacenan como un **documento de escrito**. De esto nacen los términos *montar* y *desmontar* un dispositivo. Es decir, la composición lógica se da sin la dependencia de la composición del hardware. Por ejemplo, esto quiere decir que no se depende de si la PC tiene 1, 3 o 5 discos duros para producir las unidades **c:**, **e:** o **k:**.

Siguiendo las convenciones comunes de Unix, el **sistema de archivos de Linux está organizado** en una serie de directorios estándar de propósito específico. Algunos de los más importantes son los siguientes:

/Dev	Contiene archivos que representan los dispositivos físicos de la computadora.
/etc	Está reservado para los archivos de configuración del sistema.
/lib	Contiene las bibliotecas necesarias para ejecutar los programas que residen en /bin.
/proc	Contiene archivos especiales que reciben información del kernel.
/sbin	Contiene programas a los que solo puede acceder el usuario 'root'.
/usr	Contiene los programas comúnmente utilizados por todos los usuarios y presenta una estructura con directorios /etc, /bin o /lib para esos programas.
/var	Contiene información temporal sobre la ejecución de algunos programas.
/home	Contiene los directorios de inicio de los usuarios del sistema.

Permisos. En Linux, como en otros sistemas Unix, se preserva una política de funciones de archivos. Las funciones se identifican con letras, las cuales son:

- **r:** permiso para leer el archivo
- **w:** permiso para escribir el archivo
- **x:** permiso para ejecutar el archivo
- **s:** permiso para realizar cambios del propietario del archivo



SISTEMA DE ARCHIVOS

Asimismo, cada permiso en Linux se puede aplicar a:

- Los propietarios de los archivos,
- Al grupo al que pertenece dicho propietario o
- Al resto de usuarios.

Lo anterior permite que este mecanismo de seguridad funcione perfectamente en grupos de trabajo con diferentes responsabilidades (**multiusuario**).

LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 1

- D.C.V.S.B. (2012, 15 febrero). *Input output in Linux*. Linux Management.
- Chapter 3 *Input and output management in Linux and how to get help*. (2022, 22 mayo). Java

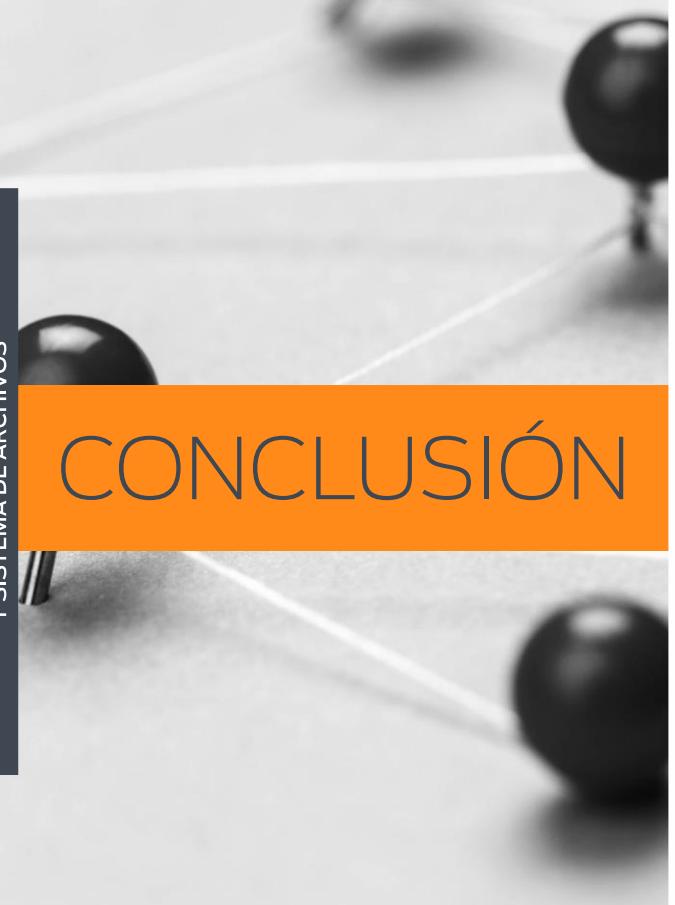
LECTURAS PARA REFORZAR LA UNIDAD



Capítulo 2

- E. (2021a, abril 3). *Sistema de Archivos de Linux: Qué es y Cómo Funciona*. BigSoftware.
- LSB Workgroup, The Linux Foundation. (2015). *Filesystem Hierarchy Standard* (Version 3.0 ed.).

CONCLUSIÓN



El control de múltiples dispositivos conectados a la computadora es un trabajo difícil. Esto se debe, principalmente, a que los dispositivos de E/S varían tanto en su funcionalidad como en su velocidad. Por ejemplo: un mouse, un disco duro y un CD necesitan de diversos métodos para ser controlados. En este sentido, dichos métodos forman el subsistema de E/S del núcleo del sistema operativo, el cual separa el resto del núcleo de las complicaciones de administrar dispositivos de E/S.

Por otro lado, aunque existen diferencias entre las distribuciones de Linux, el diseño de sus sistemas de archivos es afortunadamente similar. Tanto es así que podrías decir: “una vez que conoces uno, los conoces todos”. Así, la mejor manera de conocer el sistema de archivos es explorarlo.



¡FELICIDADES!

Acabas de concluir la **cuarta unidad** de tu curso *Sistemas Operativos I*. Te invitamos a finalizar este esfuerzo realizando el examen parcial correspondiente. Para ello, debes regresar a la pantalla principal y dar clic en *Presentar examen*.

BIBLIOGRAFÍA



- What is Linux? (2022, 6 abril).
- GeeksforGeeks. (2022, 8 marzo). Kernel in Operating System.
- Naushad, A. (2022, 5 junio). *Basic Linux Commands for Beginners*. Maker Pro.



BIBLIOGRAFÍA

- A. (2018). *UNIX / Linux : What Is a Shell? What are different Shells?* The Geek Diary.
- Loshin, P. (2021). *bash (Bourne again shell)*. SearchDataCenter.
- Garrison, J. (2017). *What is the Linux Kernel and What Does It Do?* How-To Geek.
- Vivek Ghate, S. (2011). *Operating System Concepts and Basic Linux Commands*. EDUCREATION PUBLISHING.



BIBLIOGRAFÍA

- NIAZI, R. (2022). *Linux Process Management: The Ultimate Guide*. Make Use Of.
- S. (2016). *Linux Process Management Tutorial For Developers*. Scriptcrunch.
- Verma, J. (2020). *Memory Management in Linux - How to Manage Linux Memory*. LinuxForDevices.
- A. (2020). *Tutorial: Beginners guide on linux memory management*. GoLinuxCloud.

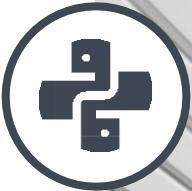


BIBLIOGRAFÍA

- Vivek Ghate, S. (2011). *Operating System Concepts and Basic Linux Commands*. EDUCREATION PUBLISHING.
- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (Fourth Edition). Pear.
- Y. (2017c, febrero 6). *¿Cómo se estructura el sistema de archivos en GNU/Linux?* Profesional Review.
- padakuu.com (2022, 26 mayo). *Linux-Input & output |*

PROYECTO FINAL





Te invitamos a realizar el proyecto final:

Presiona el botón para descargar el proyecto final:



Presiona el botón para entregar el proyecto final:

