ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Sistemas operativos:
Una exploración y aprendizaje en Linux

Walter Marcelo Fuertes Díaz
Aymé Alejandra Escobar Díaz
Ricardo Xavier Rivadeneira Gómez

Sistemas operativos: Una exploración y aprendizaje en Linux

Walter Marcelo Fuertes Díaz, Aymé Alejandra Escobar Díaz y
Ricardo Xavier Rivadeneira Gómez

Primera edición electrónica: agosto, 2026
ISBN: 978-9942-652-56-0

Ingreso al proceso: 19/03/2026 • Aprobación de ingreso: 24/04/2026
Aprobación de pares: 09/06/2026 • Finalización de edición: 16/07/2026

Universidad de las Fuerzas Armadas ESPE
Oswaldo Mauricio González Mosquera, Crnl. Ph. D.
Rector

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas ESPE
Cpfg. César Antonio Portero Barrantes, Mgtr. - Vicerrector de Investigación, Innovación y Transferencia de Tecnología (E) - Presidente

Corrección de estilo y diseño
Mtr. Xavier Chinga Mármol

Imagen de cubierta: Jairo Smith Bonilla Hidalgo (con uso de IA generativa)

Estudiantes colaboradores: Saray Adriana Cañarte Galarza, Yeshua Amador Chiliquinga Amaya, Andrés Alexander Espín Andrade, José Miguel Sanmartín Galán, Denise Noemi Rea Díaz, Julio Enrique Viche Castillo, Stefany Maricela Díaz Antún, Diego Mateo Sosa Flores, Marcos David Escobar Vela, Joyce Denise Castro Zambrano

Universidad de las Fuerzas Armadas ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador
www.espe.edu.ec

Departamento de Ciencias de la Computación

Sistemas Operativos: Una Exploración y Aprendizaje en Linux

Walter Marcelo Fuertes Díaz
Aymé Alejandra Escobar Díaz
Ricardo Xavier Rivadeneira Gómez

EDITORIAL
UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

Dedicatoria

Dedico esta obra a:
A mi amada esposa Silvia, por su amor y apoyo incondicional.
A mis queridos hijos Walter Marcelo y Mateo Alejandro, mi mayor inspiración.
A Mickaela T., por descubrir en la educación una noble vocación.
A mi padre Edmundo y a mi madre Marujita, que de Dios goce.
A Aymé y Ricardo, mis estudiantes y hoy colegas, por su dedicación y esfuerzo para culminar esta obra en conjunto.

_____ Walter

Esta obra va dedicada a:
A mi mamá Silvana, por su fortaleza, amor y ejemplo de vida.
A mi familia, especialmente a Cami y Gerald, por su cariño y apoyo constante.
Al Ing. Walter Fuertes, Ph.D., por su guía y confianza durante este proyecto.
A Ricardo, por su compromiso, dedicación y amistad, fundamentales para culminar este proyecto.
A quienes contribuyeron con su apoyo y confianza a mi formación personal y profesional.

_____ Aymé

Esta obra la dedico a:
A mi madre Jenny, por su gran amor y apoyo incondicional.
A mi padre Rafael, por su fortaleza y perseverancia, cuyo ejemplo sigue inspirándome.
A mi hermano Matias, por su compañía y apoyo constante, y a Bachita, por su amor incondicional.
Al Ing. Walter Fuertes, Ph.D., por su orientación académica y confianza.
A Aymé, por compartir este desafío con dedicación, responsabilidad y compromiso.

_____ Ricardo

Agradecimientos

Los autores expresan su sincero agradecimiento al Departamento de Ciencias de la Computación de la Universidad de las Fuerzas Armadas ESPE por proporcionar un excelente entorno académico y científico que propició este trabajo. Asimismo, reconocen el inestimable apoyo del Grupo de Investigación en Sistemas Distribuidos, Ciberseguridad y Contenido (RACKLY) y del Laboratorio de Investigación de Altas Prestaciones, cuya infraestructura tecnológica, recursos de investigación y entorno científico colaborativo contribuyeron significativamente al desarrollo y la finalización de este libro.

Prólogo

La academia en el mundo, consciente de la alta responsabilidad que conlleva cumplir con sus funciones sustantivas de docencia, investigación y vinculación con la sociedad, tiene la misión permanente de gestionar estrategias innovadoras de enseñanza-aprendizaje. El propósito fundamental es proveer a estudiantes, docentes y a la comunidad en general recursos didácticos de alta relevancia, rigor científico y vigencia técnica.

Dentro de este contexto, se presenta el libro académico titulado ***Sistemas Operativos: Una exploración y aprendizaje en Linux***. Esta obra es el resultado de un riguroso esfuerzo de compilación, investigación, experiencia docente y actualización de saberes en torno a una disciplina trascendental para la educación superior, indispensable en el currículo de las ciencias computacionales en todo el mundo.

La presente obra aborda de manera integral los fundamentos y aplicaciones de los sistemas operativos, con énfasis en entornos Linux. Su contenido desarrolla una secuencia lógica que inicia con los conceptos esenciales de la disciplina y avanza hacia temas relacionados con fundamentos de Sistemas Operativos, programación en Shell Script, virtualización, programación de procesos e hilos en lenguaje C para Linux, gestión de servicios en redes, ciberseguridad y microservicios.

El texto integra fundamentos teóricos, actividades prácticas y ejercicios de autoevaluación. Esta estructura favorece el aprendizaje progresivo de los contenidos y proporciona una guía de apoyo para estudiantes y docentes de carreras afines a las ciencias de la computación.

La incorporación de tecnologías actuales como contenedores, microservicios, servicios de red y mecanismos de seguridad amplía el alcance de la obra y establece vínculos entre los conceptos clásicos de los sistemas operativos y las tecnologías utilizadas en la actualidad.

Asimismo, el uso de Linux como plataforma principal de aprendizaje acerca al lector a herramientas ampliamente empleadas en entornos académicos, empresariales y de investigación.

En virtud de los notables atributos académicos, pedagógicos y técnicos evidenciados durante el proceso de evaluación, se concluye con certeza que esta obra constituye un aporte significativo para la formación universitaria en el área de la ingeniería, convirtiéndose en un referente de consulta obligada para estudiantes, académicos y profesionales del sector tecnológico. Finalmente, es un honor destacar la solvencia técnica y el compromiso pedagógico plasmados por los autores en este manuscrito. Respaldo con entusiasmo la publicación de esta obra, con la plena convicción de que se convertirá en una herramienta de gran beneficio e impacto para la comunidad científica y profesional que dinamiza el campo de las Tecnologías de la Información y la Comunicación

Ing. Daisy Elizabeth Imbaquingo Esparza, PhD
Docente Investigadora Titular
Subdecana de la Facultad de Ingeniería en Ciencias Aplicadas
Universidad Técnica del Norte,
Ibarra, Ecuador

Prefacio

Según StatCounter Global Stats y W3Techs, en enero de 2026 Android figura como el sistema operativo con mayor presencia en dispositivos móviles, seguido por iOS; en equipos de escritorio, Windows continúa liderando, mientras que macOS y Linux mantienen participaciones relevantes. En el ámbito de servidores y servicios web, los sistemas tipo Unix/Linux concentran la mayor parte de los despliegues, y Linux es también la base predominante en supercomputación. Además, Android se apoya en el núcleo de Linux, y macOS e iOS derivan de tecnologías Unix/BSD (Darwin). En este contexto, resulta evidente la necesidad de fortalecer el aprendizaje y uso de sistemas operativos, en particular de la familia Unix/Linux.

Desde el advenimiento de la Fundación de software libre en 1984 creada por Richard Stallman, se enfocaron los esfuerzos en crear software que no requiera Licenciamiento. Bajo este principio, Linus Torvalds en 1991, creó Linux, para que fuera estudiado, usado, modificado y distribuido libremente. Desde ese entonces, en los departamentos de Computación e Informática de las universidades, se enseña el software libre. Con esta experiencia, junto a Aymé y Ricardo, dos nóveles profesionales de la ingeniería, decidimos publicar esta obra para que sea una guía técnica de estudiantes y profesionales que tengan el interés de aprender sistemas operativos, explorando Linux.

La práctica acumulada en el uso de sistemas Unix/Linux desde la formación universitaria a nivel de ingeniería, así como su aplicación rigurosa en actividades de docencia, investigación y desarrollo, incluyendo trabajos de posgrado y doctorado, permitieron identificar la importancia de un enfoque práctico sustentado en bases conceptuales sólidas. Este enfoque constituye uno de los ejes centrales de este libro.

El libro está dirigido a estudiantes de grado, postgrado y profesionales que deseen profundizar en este tema, entendiendo que a través de los sistemas operativos se optimiza el uso del procesador, memoria, capacidad de transmisión, servicios en red y la ciberseguridad. Esta obra además está dirigida al público en general para que inicie su entrenamiento y aprendizaje.

El objetivo principal de esta obra es proporcionar una base teórica y práctica sobre las funcionalidades esenciales de los sistemas operativos, utilizando entornos virtualizados y distribuciones Linux como plataforma para las actividades experimentales y de laboratorio. Con este propósito, el contenido recoge los temas fundamentales presentes en los programas de estudio de ingeniería a nivel nacional e internacional. En consecuencia, el libro está estructurado como sigue:

- Capítulo I: Fundamentos de los Sistemas Operativos.
- Capítulo II: Tecnologías de Virtualización y su evolución.
- Capítulo III: Clasificación de Sistemas Operativos.
- Capítulo IV: Explorando Linux.
- Capítulo V: Programación en Shell Script.
- Capítulo VI: Gestión de archivos y carpetas.
- Capítulo VII: Gestión de Procesos.
- Capítulo VIII: Programación de procesos en C para Linux.
- Capítulo IX: Hilos.
- Capítulo X: Algoritmos de Planificación de Procesos.
- Capítulo XI: Gestión de memoria interna y de almacenamiento.
- Capítulo XII: Gestión de servicios de red en Linux.
- Capítulo XIII: Gestión de ciberseguridad en Sistemas Operativos.
- Capítulo XIV: Microservicios y Contenedores en Linux.

Esperamos que disfruten revisando esta obra y que estos conocimientos sean en beneficio de la sociedad a la cual nos debemos.

Los autores

Walter Fuertes
wmfuertes@espe.edu.ec

Es Ingeniero de Sistemas e Informática y magister en Docencia Universitaria, títulos que los obtuvo en la Escuela Politécnica del Ejército. Asimismo, se graduó de magister en Informática mención Redes, por la Escuela Politécnica Nacional. Ambas universidades de Quito, Ecuador. Obtuvo su grado de Doctor en Ingeniería Informática y de Telecomunicaciones con honores, en la Universidad Autónoma de Madrid, España. En su trayectoria ha desarrollado varios proyectos de investigación, ha escrito más de un centenar de artículos científicos indexados en Scopus, ha publicado tres libros técnico-académicos y ha sido expositor en varios congresos nacionales e internacionales. Sus intereses de investigación están enfocados a la seguridad de la información, ciberseguridad, ciencia de datos e inteligencia artificial.

Aymé Escobar Díaz
aaescobar2@espe.edu.ec

Graduada en la Universidad de las Fuerzas Armadas ESPE, es una ingeniera en Software con experiencia en el desarrollo de aplicaciones de tipo ERP y agentes inteligentes, así como en la configuración, administración de sistemas Linux en ambientes de producción. Ha trabajado con arquitecturas basadas en microservicios y en la automatización de procesos en entornos de producción.

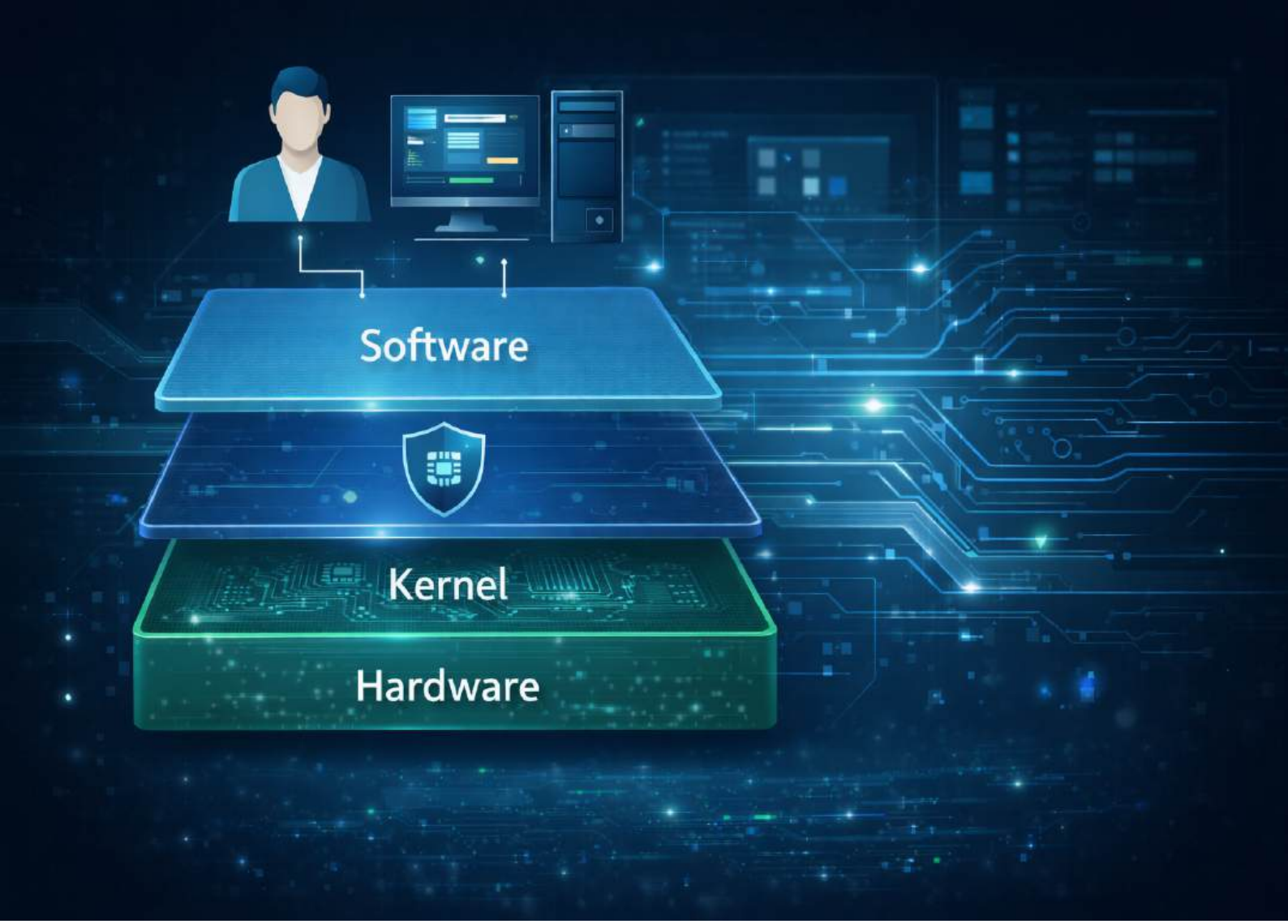
Es autora y coautora de trabajos científicos relacionados con la detección y mitigación de ataques de ciberacoso y el tono en las palabras de odio en contra de la mujer en el ciberespacio. Es amante del la analítica de datos utilizando algoritmos de aprendizaje automático y de procesamiento de lenguaje natural.

Ricardo Rivadeneira
rxrivadeneira1@espe.edu.ec

Es ingeniero de software con experiencia en el desarrollo de aplicaciones móviles y plataformas web, así como en la administración de sistemas Linux y el diseño de soluciones tecnológicas orientadas a entornos empresariales. Ha participado en proyectos relacionados con la automatización de procesos y la integración de sistemas de información para la gestión organizacional.

Es autor y coautor de trabajos científicos relacionados con la ciberseguridad, detección y mitigación del ciberacoso y el análisis del discurso de odio hacia la mujer en el ciberespacio. Sus intereses profesionales se centran en el desarrollo de software, las tecnologías emergentes, los sistemas operativos y las aplicaciones informáticas orientadas a la gestión financiera y empresarial.

Índice



CAPÍTULO I

Fundamentos de los Sistemas Operativos

Introducción

Con el avance acelerado de las tendencias tecnológicas, tales como la Industria 4.0, la interacción humano-computadora (Human-Computer Integration, (HCI)) por sus siglas en inglés, la Inteligencia Artificial, la ciudadanía digital, las redes, las telecomunicaciones, y la computación en la nube, el usuario, para su vida diaria, requiere el empleo de una computadora, un teléfono inteligente, un dispositivo de conectividad, o cualquier otro aparato que funcione dentro de la red Internet. Para este propósito, el usuario debe aprender a utilizar y/o configurar los ***Sistema Operativos*** que vienen embebidos en ellos y que sirven para poder aprovechar los servicios y aplicaciones.

Los ***Sistemas Operativos*** en esencia son aquellas piezas de software que vienen preinstaladas en todos los equipos y dispositivos para permitir su funcionamiento y además para facilitar la gestión, uso y aprovechamiento. Sin este software, no sería posible la configuración de servicios, la instalación de aplicaciones, la gestión de sus recursos tales como la CPU, memoria interna, de almacenamiento, y dispositivos de entrada y salida.

Este primer capítulo aporta en la formación del lector porque debe conocer cómo el usuario interactúa con el software, el hardware y los recursos computacionales. Así mismo, porque necesita comprender cómo interactúan sus programas con el ***Sistemas Operativos*** subyacente.

Al finalizar este capítulo, el lector comprenderá los fundamentos básicos de los ***Sistemas Operativos***, aprenderá a definirlos, entenderá sus objetivos y funciones, cómo ha evolucionado en una línea de tiempo desde sus inicios en 1960 hasta la presente fecha. Aprenderá a diferenciar entre sistemas de código abierto o libre distribución versus sistemas propietarios o comerciales. Así mismo, identificará algunos personajes como Bill Gates, Richard Stallman y Linus Torvalds quienes originaron avances trascendentales en el desarrollo de los mismos, su historia, filosofía y principales obras.

En este capítulo el lector además encontrará actividades de aprendizaje, prácticas de laboratorio, problemas propuestos, resueltos, y sitios de interés que facilitarán la apropiación de su conocimiento.

Como plataforma de entrenamiento que permitirá realizar las pruebas de concepto y formalizar el aprendizaje se utilizará una distribución de Linux tanto para desktop como para servidor.

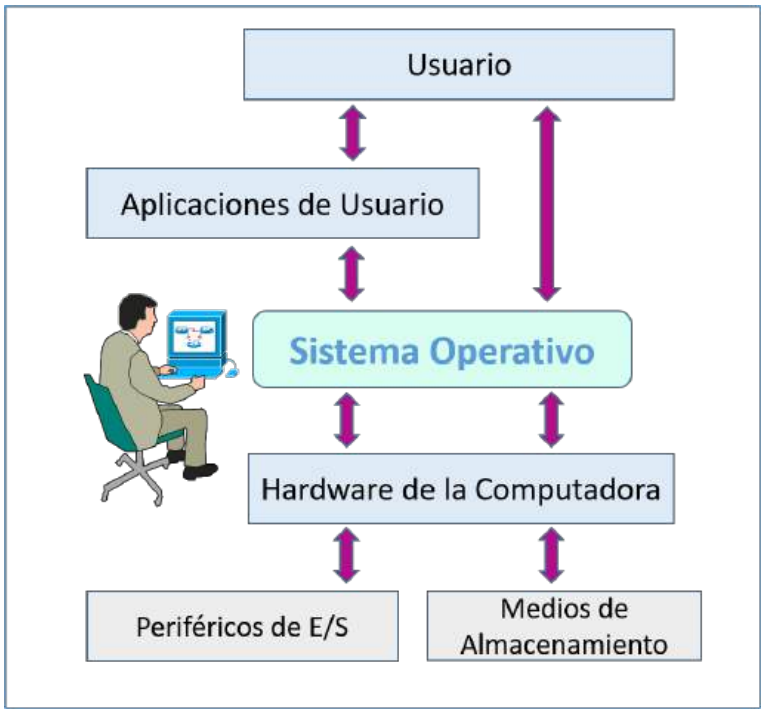
¡Bienvenidos!

Definición de sistema operativo

De acuerdo con Tanembaum [127] un ***Sistema Operativo*** es el software que permite la interacción entre el hardware de una computadora y los usuarios. Cumple con funciones esenciales, como suministrar la interfaz al usuario, administrar recursos, archivos, tareas, y brindar soporte, aplicaciones y servicios. Se encarga de gestionar los recursos del sistema, tales como la memoria, el procesador y los dispositivos de entrada/salida.

Un ***Sistema Operativo*** es un conjunto de programas en un sistema informático que administra recursos de hardware y proporciona servicios a programas de aplicaciones de software [133]. En otras palabras, es el puente o medio de comunicación entre el usuario y la computadora. Se ocupa de la carga y ejecución de programas de aplicación y gestiona la transferencia de datos y archivos hacia y desde los dispositivos periféricos [31]. Además, es el responsable de la sincronización, gestión de seguridad y conectividad en redes de computadoras, de telefonía celular y de telecomunicaciones [66]. La Figura I.1 es una representación visual de cómo interactúa el ***Sistema Operativo*** con el hardware de la computadora, las aplicaciones y el usuario:

Figura I.1: Gráfica conceptual de un sistema operativo. El S.O interactúa como un puente entre las aplicaciones del usuario y el hardware.



Para facilitar la comprensión del lector, considere un programa de software que requiere que el usuario ingrese su contraseña. El ***Sistema Operativo*** da acceso al usuario al disco donde se almacena el programa. Además accede a la memoria del dispositivo para cargarlo y así poder ejecutarlo. Asimismo, permite mostrar al usuario cómo ingresar su contraseña mediante mensajes en la pantalla, habilitando al teclado y al ratón (mouse) para que el usuario introduzca su contraseña. Finalmente, permite al programa especializado leer y validar su contraseña, para que lo pueda utilizar.

Existe una variedad de ***Sistemas Operativos*** para equipos de escritorio: (Windows, Linux, Mac OS); para servidor: (Windows Server, UNIX, Solaris, AIX, Linux Server), para teléfonos móviles: (Android, IOS, Windows phone, BlackBerry OS), para dispositivos de red: (Cisco IOS, OpenSwitch de HP, FortiOS, de Fortinet) por citar algunos ejemplos [2], [127]. La Figura I.2 presenta los sistemas operativos más utilizados en el mundo.

Figura I.2: Los Sistemas Operativos más utilizados en el mundo. Fuente: Esta ilustración fue creada por los autores utilizando Copilot IA

Esta variedad permite a los usuarios seleccionar el sistema operativo que mejor se adapte a sus necesidades y preferencias individuales o empresariales, lo que mejora su experiencia general con la tecnología [2], [127]. La Figura I.3 y I.4 muestran la interfaz de cada sistema operativo para un smartphone. Cabe destacar que en un smartphone existen controladores para la cámara fotográfica, la video cámara, linterna, Internet, WIFI, telefonía celular, entre otros servicios y aplicaciones, mismos que son gestionados por el sistema operativo.

Figura I.3: *Interfaz de Iphone.*
***Fuente**: www.pixabay.com*

Figura I.4: *Interfaz de Android.*
***Fuente**: www.pixabay.com*

Con el propósito de facilitar la comprensión de la estructura interna de un sistema operativo, la Figura I.5 muestra la arquitectura e interacción por capas de un sistema operativo Linux. En ella se observa cómo el usuario interactúa con las aplicaciones, las cuales acceden a los servicios del sistema operativo a través del Shell y las llamadas al sistema (System Calls). Estas solicitudes son procesadas por el Kernel, que administra los recursos del hardware mediante controladores de dispositivos (Drivers). Esta organización por capas permite abstraer la complejidad del hardware y proporciona una interfaz estandariza-

da para el desarrollo y ejecución de aplicaciones.

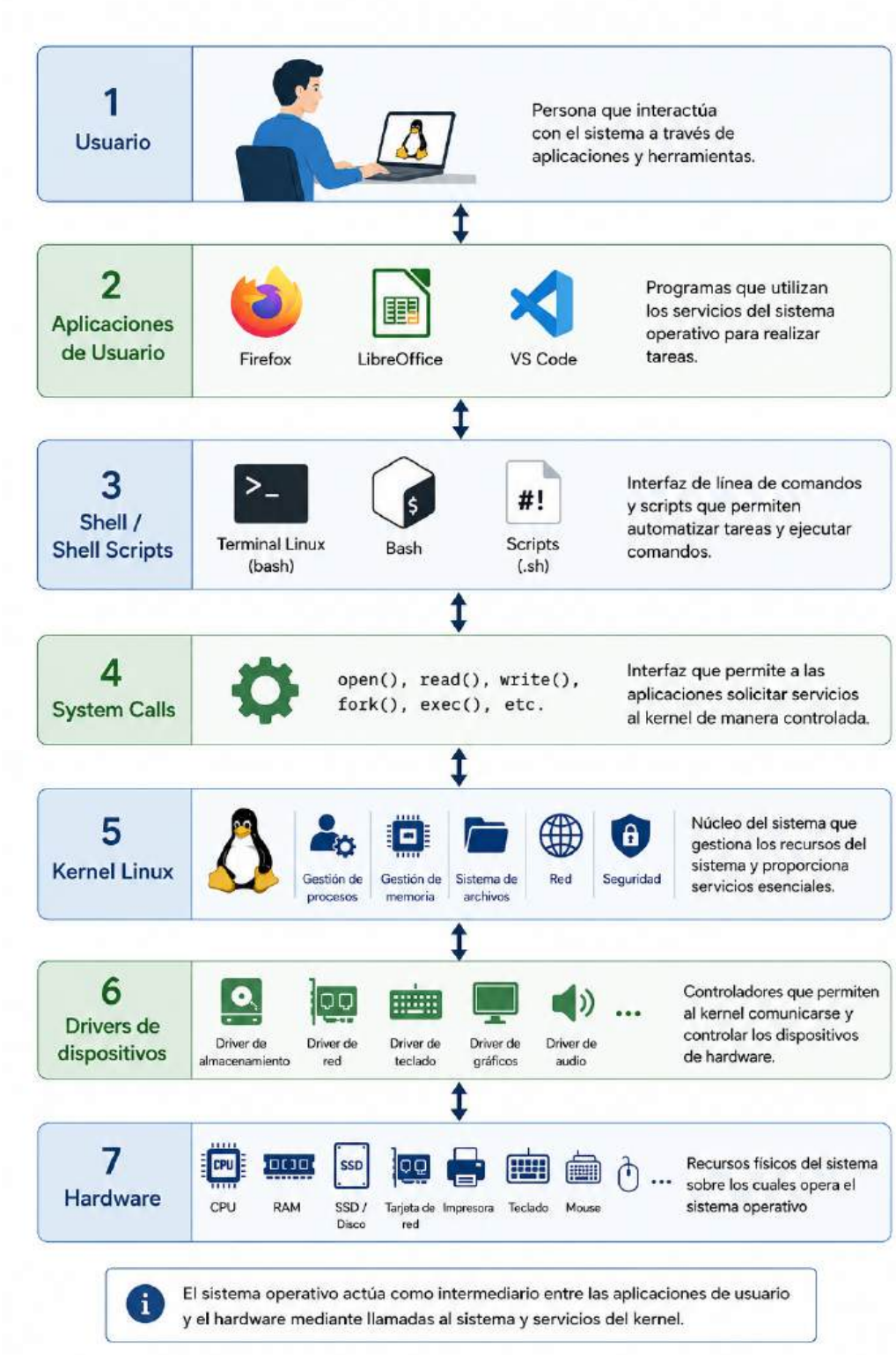


Figura I.5: *Arquitectura e interacción por capas de un sistema operativo Linux.*
Fuente: Esta ilustración fue creada por los autores utilizando ChatGPT.

Funciones de un sistema operativo

Los ***Sistemas Operativos*** son la parte central de cualquier dispositivo informático. Su función esencial es administrar recursos y permitir que los usuarios interactúen con el hardware y el software. Estas funciones son orientadas a incrementar la eficiencia en el uso de recursos, la seguridad, la conectividad y la usabilidad, y son cruciales para comprender su diseño y funcionamiento [66]. En esta breve exploración, se examinarán las principales funciones, como plataforma que da soporte a los servicios y aplicaciones requeridos por el usuario:

Abstracción del hardware: El ***Sistema Operativo*** proporciona una capa de abstracción que oculta los detalles técnicos del hardware al usuario final. Esto permite que el usuario interactúe con el sistema a través de un conjunto de aplicaciones sin necesidad de conocer los aspectos internos del hardware [119]. Para cumplir este propósito, el sistema operativo provee al usuario de una interfaz para gestionar su equipo. Las ampliamente utilizadas se pueden distinguir en la Figura I.6 y se describen brevemente a continuación:

Figura I.6: Diversos tipos de interfaz de usuario en los Sistemas Operativos. Fuente: Esta ilustración fue creada por los autores utilizando Copilot IA

- Interfaz de línea de comandos (Command Line Interface, (CLI)) que proporciona una interfaz basada en texto en modo terminal. Permite ingresar por teclado comandos, parámetros o datos específicos.

- Interfaz gráfica de usuario (Graphical User Interface, (GUI)) utilizada con mayor frecuencia, que proporciona una interfaz visual basada en ventanas, iconos y símbolos en la que los usuarios ingresan los datos específicos, vía ratón, teclado, e inclusive paneles y pantallas táctiles.

- Interfaz basada en voz (Voice User Interface, VUI), que proporciona una interfaz basada en voz. Permite que a través del sonido se impartan órdenes o comandos. Ejemplos: Alexa, Google Assistant, Siri, e inclusive IA Generativa [91].

- Interfaz basada en gestos (Gesture-Based Interface, GBI) que son tecnologías que permiten reconocer movimiento utilizando gestos físicos para interactuar con el sistema operativo [38].

Las interfaces previamente señaladas permiten a los usuarios y administradores interactuar con el ***Sistema Operativo*** para instalar, personalizar, configurar, depurar, probar e incluso resolver problemas (troubleshooting) del equipo, hardware y software subyacente [127]

Arranque del Sistema computacional: Al prender e iniciar el computador o cualquier dispositivo, el sistema operativo se encarga de verificar los recursos del dispositivo para activarlos. Este se conoce como el proceso de arranque, en el cual todos los dispositivos encontrados son inicializados para que el sistema computacional funcione.

Ofrece una variedad de aplicaciones Para facilitar el desarrollo de software, el ***Sistema Operativo*** ofrece una variedad de herramientas y servicios, como editores, lenguajes de programación, ilustradores y depuradores, que ayudan al programador en la implementación de sus programas. Además, se encarga de realizar una serie de tareas para ejecutar programas, como cargar instrucciones y datos en la memoria principal, inicializar dispositivos de E/S y preparar recursos necesarios

Gestión de almacenamiento El ***Sistema Operativo*** permite acceder o almacenar los datos e información del usuario en los procesos de lectura y escritura al disco duro o sólido. Además, permite almacenar las aplicaciones o programas de software requeridos para su vida diaria.

Gestión de recursos computacionales: Al asignar recursos como la memoria principal, el tiempo de procesador o los dispositivos de E/S, le permite al Sistema Operativo recopilar estadísticas de uso de recursos y monitorear el rendimiento del sistema, para mejorar o asegurar el desempeño del sistema computacional.

Gestión de archivos y sistemas de archivos: Componente principal de los Sistemas Operativos, se utiliza para crear, manipular, almacenar y recuperar archivos en el sistema de almacenamiento [70], asimismo, para la estructura y organización de los datos en disco.

Gestión de procesos y tareas múltiples: Implica la creación, ejecución, suspensión y finalización de procesos, a la vez que gestiona recursos del sistema [98]. Permite que múltiples programas se ejecuten simultáneamente y compartan los recursos del sistema, donde se mejora el rendimiento general del sistema.

Seguridad y protección: Garantizan la seguridad del sistema y protegen los datos y recursos del usuario mediante la implementación de mecanismos de control de acceso, autenticación, cifrado y detección de amenazas. Los sistemas operativos tienen la responsabilidad de ofrecer soluciones de hardware y software para proteger la integridad, disponibilidad y confidencialidad de la información del usuario e impedir accesos no autorizados al sistema computacional.

Gestión de errores y tolerancia a fallos: Es la capacidad del sistema para continuar su funcionamiento sin interrupciones cuando falla uno o más componentes [96], proporcionan mecanismos para detectar, diagnosticar y recuperarse de errores de manera que minimiza el impacto en la integridad y disponibilidad de los datos.

Gestión de la conectividad en redes El Sistema Operativo es el responsable de la configuración de redes de datos en el equipo o dispositivo. Se incluyen la gestión de direccionamiento IP o lógico, direccionamiento físico o de hardware, la dirección de pasarela, redes privadas virtuales, gestión de las interfaces de red, la interconexión a Internet, entre otras.

Evolución de los sistemas operativos

Desde el aparecimiento de la informática, la microelectrónica, la Inteligencia Artificial, las telecomunicaciones y el Internet, la industria ha enfrentado desafíos cada vez más complejos. Esta realidad ha impulsado la evolución y

desarrollo de los sistemas operativos, que actualmente se enfocan en la usabilidad, accesibilidad y experiencia del usuario. Este desarrollo ha generado una serie de sucesos tecnológicos que deben ser conocidos por el lector. Al examinar esta evolución, se comprueban aportaciones esenciales que han modelado la informática en el trascurrir del tiempo.

Dentro de este contexto, se puede destacar que esta evolución está estrechamente ligada a la arquitectura de las computadoras que los soportan a lo largo de los años [24]. Este progreso se ha caracterizado por mejorar la manera en que los sistemas gestionan los recursos y cómo interactúan con los usuarios. Desde los sistemas batch iniciales (i. e., trabajo por lotes) hasta las interfaces gráficas modernas [79].

Para ayudar con la compresión, la evolución de los sistemas operativos ha sido clasificada por generaciones con relación a la evolución del hardware y los sistemas operativos [25]. A continuación, se presentan los avances y cambios en la tecnología, con respecto al progreso en el hardware y las demandas de los usuarios.

Primera Generación de sistemas operativos (1945 - 1955)

Se utilizaron tubos de vacío y tableros enchufables. Durante ese período, las tareas se agrupaban en lotes y se ejecutaban una tras otra, con cada tarea controlando completamente la máquina. Una vez finalizada una tarea, el control volvía al sistema operativo, que limpiaba, leía e iniciaba la tarea siguiente [59].

El primer sistema operativo IBM 701, se adjudica a los laboratorios de investigación de General Motors [132]. En 1950, salieron a la luz las tarjetas perforadas, lo que supuso una mejora en la escritura y la lectura de los datos, ya que se dejó de lado los tableros enchufables, pero el proceso general se mantuvo de igual forma.

Transistores y sistemas de procesamiento por lotes - (1955-1965)

Durante este período, hubo avances significativos en la tecnología de las computadoras con la introducción de los transistores. Estos componentes electrónicos reemplazaron a los antiguos tubos de vacío, lo que permitió una mayor miniaturización y eficiencia en los sistemas computacionales.

Surgieron los sistemas de procesamiento por lotes, los cuales automatizaban la ejecución de programas sin intervención manual del usuario, es decir,

las tareas se agrupaban en lotes y se ejecutaban secuencialmente.

Circuitos integrados y multiprogramación - (1965-1980)

Durante la Tercera Generación de sistemas operativos, marcada por la aparición de los circuitos integrados y la multiprogramación, se desarrollaron los computadores IBM/360. Esto llevó a una evolución a los sistemas de modos múltiples, que podían soportar procesos por lotes, tiempo compartido, procesamiento en tiempo real, multiprocesamiento y multiprogramación, que permitía la ejecución simultánea de múltiples programas. Esta mejora significativa mejoraría la eficiencia del uso de los recursos al permitir que la CPU alternara entre diferentes tareas de manera rápida y paralela, debido a que la CPU podía ejecutar varios programas en paralelo.

Computadoras personales - (1980-actualidad)

El surgimiento de las computadoras personales en la década de 1980 revolucionó la informática al poner el poder de la computación en manos de los usuarios individuales. Esto llevó al desarrollo de sistemas operativos diseñados específicamente para satisfacer las necesidades y preferencias de los usuarios domésticos y empresariales.

MS-DOS y Windows de Microsoft junto con MacOS para PCs, y Unix para servidores, durante los años 70 al 90 fueron los más empleados por las empresas, en razón del inicio de la Informática en el mundo, la multitarea, el multiprocesamiento, video juegos y navegación en el ciberespacio.

Con el tiempo, los sistemas operativos para computadoras personales evolucionaron para incluir interfaces gráficas de usuario (GUI) intuitivas y amigables.

Esta generación estuvo impulsada por la aparición de las computadoras personales conectadas en red. De esa manera, y con la expansión del uso de redes de computadoras y el procesamiento en línea, surgió la oportunidad de acceder a computadoras remotas a través de diferentes tipos de terminales, servidores, y nubes computacionales.

Generación de nuestros días

Para analizar la actualidad de la evolución de los sistemas operativos se podría realizar la siguiente clasificación:

- **Para equipos de escritorio:** Microsoft Windows históricamente ha sido un monopolio. Un listado de su evolución y las características más destacadas se listan en la Tabla I.1. En este mismo criterio, conviene realizar el análisis de su competencia, es decir, el sistema operativo macOS. Cabe señalar que desde 1991 existen diversas distribuciones de Linux como Debian, Ubuntu, Fedora, Centos, Red Hat, Caldera, Kali Linux, entre otras, que son utilizadas tanto para su uso como cliente y servidor en todo el mundo. Estas distribuciones pertenecen a la filosofía de software libre y disponen de soporte técnico constante. Sin embargo, estos no han podido superar el monopolio establecido por Microsoft Windows en equipos de escritorio. En los próximos apartados se realizará una descripción detallada de Linux y servirá como plataforma de aprendizaje en el contexto de este libro.

- **Para equipo servidor:** Microsoft Windows Server es ampliamente utilizado en empresas públicas y privadas, especialmente para dar eficiencia los modelos de negocio en red, la gestión de perfiles y cuentas de usuario. Estos sistemas son de uso propietario, comercial o con pago de licenciamiento. Cabe señalar que los sistemas operativos Unix, AIX, Solaris, y Linux realizan tareas y funciones similares, aunque adicionalmente se le atribuye la gestión de seguridad y conectividad de empresas con un número grande de usuarios.

- **Para dispositivos móviles:** IOS y Android son los principales para el desarrollo de aplicaciones. Originalmente fueron creados para recibir llamadas telefónicas. Luego se le añadió el envío de mensaje de textos. Dado el desarrollo del Internet, se le incorporó el envío y recepción de correos electrónicos, videos en YouTube, videoconferencias, cámara de fotos y videos e inclusive sensores, actuadores y mecanismos de expansión.

Tabla I.1: Evolución de Microsoft Windows para escritorio

Versión	Año	Características principales
Windows 1.0	1985	Interfaz gráfica basada en ventanas, corría sobre el sistema operativo MS-DOS
Windows 2.0	1987	Mejora de gráficos, memoria ampliada, iconos de escritorio y un panel de control
Windows 3	1990	Mejoras en rendimiento y multitarea, renovación de los iconos del escritorio
Windows NT	1993	Sistema operativo de 32 bytes multitarea, multiusuario y multiprocesamiento
Windows 95	1995	Nueva interfaz gráfica y de usuario, navegador Internet Explorer, botón de inicio, barra de tareas y el área de notificaciones
Windows 98	1998	Primera versión diseñada específicamente para el consumidor, nuevo sistema de archivos más rápido y con más capacidad
Windows Millennium Edition	2000	Orientado al mercado profesional, recibió críticas debido a sus problemas de estabilidad y seguridad
Windows XP	2001	Rediseño espectacular, estética limpia, funcionamiento eficaz y con pocos errores
Windows Vista	2006	Mejora en la seguridad y en la interfaz de usuario, pero fue criticado por su rendimiento lento y problemas de compatibilidad
Windows 7	2009	Mejora en el rendimiento y la estabilidad, introducción de la barra de tareas rediseñada y nuevas funciones de interfaz de usuario
Windows 8	2012	Interfaz de usuario completamente rediseñada orientada a dispositivos táctiles, introducción de la pantalla de inicio
Windows 10	2015	Retorno del menú de inicio, introducción de Cortana, actualizaciones gratuitas para usuarios de Windows 7 y 8
Windows 11	2021	Rediseño completo de la interfaz, centrado en la simplicidad y la productividad, introducción de Microsoft Teams integrado

Parte de esta información fue recopilada de: Omega2001 - La Evolución de Windows, Marzo de 2021. Disponible en:
`https://omega2001.es/la-evolucion-de-windows-1985-2021/`

Sistemas Operativos de código abierto o libre distribución

De acuerdo con Mannila [83] los sistemas operativos de código abierto, surgieron debido la filosofía del software libre, mejor conocida por su patrocinio al proyecto GNU iniciado por Richard M. Stallman en 1980, y se distinguen por su transparencia y libertad de uso. Esta filosofía defiende la libertad del usuario para utilizarlo, estudiarlo, modificarlo y redistribuirlo de ser el caso [125]. Stallman propuso la creación de la Fundación de Software Libre (Free Software Foundation, FSF), como una organización sin fines de lucro que tiene como objetivo principal promover la libertad de los usuarios de software. La FSF es conocida además por su papel en la redacción de la Licencia Pública General de GNU (GPL), que garantiza a cualquier desarrollador el acceso al código fuente del software para que pueda explorar cómo funciona y realizar modificaciones de ser necesario.

Por otro lado, la FSF fue establecida debido a la restricción de libertades en el uso del software propietario, puesto que las empresas imponían costos a la copia, modificación y distribución de programas informáticos. Stallman y otros activistas como Bruce Perens y Robert Chassell crearon la FSF con la visión de construir un movimiento que promoviera la libertad del software, enfocándose en la creación y promoción de software libre, que permite a los usuarios disfrutar de estas libertades fundamentales [125]. A continuación, se describen las cuatro libertades del software libre, y una leve explicación del Proyecto GNU-Linux:

- **Libertad 0:** Implica la capacidad de utilizar el programa con cualquier objetivo;
- **Libertad 1:** Implica la posibilidad de examinar y modificar el programa;
- **Libertad 2:** Otorga la libertad de copiar el programa de manera libre, lo que incluye su redistribución;
- **Libertad 3:** Brinda a los usuarios la facultad de mejorar el programa e incorporar esas mejoras a nuevas versiones.

GNU/Linux

Linux es un de los mayores ejemplos que se puede dar a la hora de hablar de un software libre, ya que como tal, permite estudiarlo, copiarlo, modificarlo

y redistribuirlo. Linux fue programado por Linus Torvalds en la década de 1990. La combinación de Linux con la filosofía del movimiento del software libre y su compatibilidad con el sistema operativo GNU dio lugar al Proyecto GNU/Linux. La Figura I.7 muestra el ícono o símbolo de Linux.

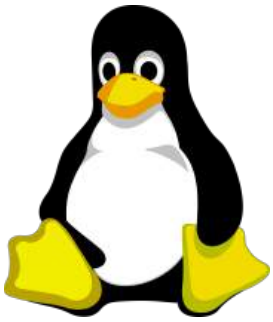


Figura I.7: Tux es el icono y símbolo de Linux. **Fuente**: www.pixabay.com

Cabe señalar que Linux (i.e., la combinación de Linus + Unix), es el nombre que se le asignó al kernel encargado de que el sistema operativo funcione. Por tanto, se destaca que varias de sus características y funcionalidades fueron heredadas de los sistemas Unix, conforme lo señala Arturo Fernández en su libro. [44]

Asimismo, la flexibilidad de GNU/Linux ha dado lugar a diversas distribuciones como Debian, Ubuntu, Fedora, Red Hat, Suse, Mandriva, Gentoo, Centos, Slackware, entre otras. Cada una presenta diferencias en aspectos como el funcionamiento interno, la apariencia visual y el método de instalación de programas, lo que permite a los usuarios encontrar la distribución que mejor se ajuste a sus necesidades y cambiar entre ellas sin inconvenientes. La Tabla I.2 lista los sitios web oficiales de algunas distribuciones de Linux.

Tabla I.2: Principales Distribuciones de Linux

Distribución	Enlace en la Web
Debian	`https://www.debian.org/`
Fedora	`https://fedoraproject.org/es/`
Ubuntu	`https://ubuntu.com/`
Kali Linux	`https://www.kali.org/`
Mandriva	`https://mandriva.com/`
Suse	`https://www.opensuse.org/`
Gentoo	`https://www.gentoo.org/`
Slackware	`http://www.slackware.com/`
MEPIX	`https://mepis.org/`
Knoppix	`https://www.yangonhotels.org/knoppixcom/`
FreeBSD	`https://www.freebsd.org/`
DistroWatch	`https://distrowatch.com/`

Arquitectura de Linux

Linux fue diseñado mediante una arquitectura de núcleo monolítico, que proporciona un rendimiento eficiente y acceso directo a los recursos del sistema. Su núcleo incluye componentes como el gestor de memoria, el programador de tareas y los controladores de dispositivos, que facilitan la gestión de recursos y la comunicación con el hardware del sistema. La Figura I.8 ilustra sus cuatro elementos: (i) El **núcleo o kernel** que interactúa directamente con el hardware; (ii) **La shell** con su dos funciones principales, encargada de la interfaz de línea de comandos (CLI) y el lenguaje de programación intérprete denominado shell script; (iii) El **sistema de archivos** que es la arquitectura de datos en donde se almacenan las librerías requeridas para su funcionamiento y los datos del usuario; (iv) **Las Aplicaciones**, que son los programas de software que utiliza el usuario en su jornada diaria.

Figura I.8: *Arquitectura del sistema*

1. **Núcleo/Kernel:** Permite interactuar con el hardware.
2. **Shell:** Lenguaje de programación, Command Line Interface.
3. **Sistema de archivos:** Arquitectura de datos, directorios, subdirectorios que se necesitan para utilizar.
4. **Aplicaciones de software:** Son una diversidad de programas de computadoras, implementadas por programadores, para hacer más eficiente el desempeño de usuarios y empresas utilizando Linux.

Arquitectura de Linux Linux es un sistema operativo de código abierto creado por Linus Torvalds y liberado en 1991.
Kernel: Linux está basado en una arquitectura monolítica, lo que significa que sus funcionalidades como la gestión de procesos, memoria, archivos, controladores de dispositivos, entre otros, se ejecutan en el espacio del kernel.
Interfaz de Usuario: Linux dispone de una variedad de interfaces de usuario, incluyendo entornos de escritorio como el Entorno de Modelo de Objeto de Red GNU (GNU Network Object Model Environment, GNOME), Entorno de Escritorio K (K desktop Environment, KDE), Entorno común de XForms (XForms Common Environment, Xfce), así como la Interfaz de Línea de Comandos (Command Line Interface, CLI).
Sistema de Archivos: Linux es compatible con varios sistemas de archivos, como el Sistema de Archivos Extendidos (Extensible File System, Ext), ext2, ext3, ext4, el sistema de archivos B-TREe (B-TRee File System, Btrfs), dependiendo de la distribución y las necesidades del usuario.

Sistemas Operativos Comerciales

Los sistemas operativos comerciales o propietarios no permiten que los usuarios vean ni modifiquen el sistema interno. La principal diferencia con el software libre, son las restricciones y costos de adquisición. Dicho valor se encuentra relacionado con el soporte técnico y las actualizaciones proporcionadas por la empresa desarrolladora, la cual mantiene control exclusivo sobre el sistema operativo para proteger la propiedad intelectual y sus intereses comerciales.

Estos sistemas operativos son los que con mayor frecuencia vienen preinstalados en los dispositivos informáticos modernos. Hershel Gonzales (2021) los define como aquellos creados por una compañía que ofrece su distribución y soporte a cambio de un pago [55].

Cabe mencionar que, a pesar de sus costos de adquisición, los sistemas operativos propietarios son los que mayor se utilizan por los usuarios, debido en parte al monopolio existente en la venta de equipos informáticos. Asimismo, el usuario debe cubrir costos de actualización para evitar errores de compatibilidad. Algunos ejemplos de sistemas operativos propietarios son los desarrollados por Microsoft, como Microsoft Windows en sus versiones para cliente y servidor.

Microsoft Windows

Desarrollado por Microsoft Corporation, Windows se ha adaptado a las diferentes demandas del mercado, como se observa en la Tabla I.1. La Figura I.9 muestra el logo de Microsoft Windows. Entre sus características representativas se encuentran las siguientes:

- Compatibilidad con una amplia gama de hardware y software de terceros;
- Brindar soporte para multitarea y multi-sesión;
- Soporte, actualizaciones regulares de seguridad y funcionalidad.

Figura I.9: El logo de Microsoft Windows. **Fuente**: www.pixabay.com

Arquitectura del sistema operativo Windows
Windows es el sistema operativo estándar creado por Microsoft para computadoras domésticas y de negocios. Dispone de una interfaz gráfica de usuario (GUI) y fue introducido en 1985.
Kernel: El sistema operativo Windows se basa en el Kernel de Nueva Tecnología (NT), encargado de la gestión de recursos como la memoria, los procesos y los dispositivos de entrada/salida [118].
Interfaz de Usuario: Windows utiliza el sistema Fundación de Presentación de Ventanas (Windows Presentation Foundation, WPF) para proporcionar una interfaz gráfica [138].
Sistema de Archivos: Windows utiliza el sistema de archivos NTFS para la organización y gestión de datos en los discos [134]. Cabe señalar que en versiones anteriores se empleaban FAT32 y exFAT, que resolvían las limitaciones de FAT32 en cuanto a tamaño máximo de archivo y capacidad de la unidad [81].

macOS

Es un sistema operativo exclusivo de los productos de la compañía, desarrollado por Apple Inc. Su carácter propietario le permite a Apple generar ingresos mediante su distribución en laptops, equipos de escritorio y dispositivos móviles [115]. La Figura I.10 muestra el logo de macOS. MacOS se caracteriza por:

- **Su integración con el entorno Apple:** debido a que macOS convive con iPhone, iPad, iPod, y Apple Watch en sus aplicaciones de Handoff , Air Drop y compatibilidad con iCloud.
- **Seguridad y protección de datos:** Incorpora funciones como Gatekeeper, que impide la instalación de software no verificado, y FileVault, que cifra el disco completo para proteger la información del usuario [17, 5].

Figura I.10: Logotipo de macOS. **Fuente**: https://icons.veryicon.com

Arquitectura del sistema operativo macOS
macOS es el sistema operativo propietario desarrollado por Apple Inc. para sus computadoras personales. Está basado en el sistema operativo Unix y combina estabilidad, seguridad y una interfaz gráfica centrada en la experiencia del usuario [115].
Kernel: macOS utiliza el kernel XNU (X is Not Unix), que combina un microkernel Mach con componentes de BSD. Se encarga de la gestión de procesos, memoria y dispositivos, proporcionando alta estabilidad y compatibilidad con estándares POSIX [118].
Interfaz de Usuario: La interfaz gráfica de macOS, denominada Aqua, está diseñada para ofrecer una experiencia visual elegante y minimalista. Incorpora elementos como el Dock, Mission Control y Finder, que facilitan la interacción del usuario con el sistema.
Sistema de Archivos: Desde 2017, macOS emplea el Apple File System (APFS) como sistema de archivos predeterminado, optimizado para unidades de estado sólido (SSD). APFS ofrece mejor seguridad, cifrado nativo, administración de snapshots y mayor eficiencia en el manejo de espacio en disco [58].

Personajes destacados en la historia de los sistemas operativos

A lo largo de la historia de la informática, varios pioneros han contribuido de al desarrollo de los sistemas operativos. Entre las figuras más influyentes se encuentran:

Dennis Ritchie

Dennis Ritchie (Bronxville, Estados Unidos; 9 de septiembre de 1941 – 12 de octubre de 2011) fue un informático estadounidense, creador del lenguaje de programación C en 1972, del cual derivan numerosos lenguajes actuales como Java, C++, C#, Swift, Python, Ruby y PHP [73]. Junto a Ken Thompson, en los Laboratorios Bell de AT&T, desarrolló el sistema operativo **Unix**, caracterizado por ser multiusuario, multitarea e interoperable. Unix constituye la base de sistemas modernos como macOS, Solaris, Linux, iOS y Android [78].

Richard Stallman

Richard Stallman (Manhattan, Nueva York; 16 de marzo de 1953), conocido por sus iniciales **RMS**, es un programador y activista estadounidense, fundador en 1984 del **Proyecto GNU** y de la **Free Software Foundation** (FSF). Su objetivo fue devolver a los usuarios la libertad de usar, modificar y redistribuir el software mediante el sistema operativo **GNU**, acrónimo recursivo de *"GNU's Not Unix"* [23], con la intención de devolver a los usuarios de computadoras la libertad que la mayoría había perdido. Su trayectoria ha sido ampliamente reconocida, destacando su influencia en el movimiento global de software libre [46], lo que significa que todos tienen la libertad de copiarlo, redistribuirlo y realizar cambios, ya sean grandes o pequeños.

Linus Torvalds

Linus Torvalds (Helsinki, Finlandia; 28 de diciembre de 1969) es un ingeniero de software reconocido por iniciar el desarrollo del kernel del sistema operativo **Linux** en 1991, durante sus estudios en la Universidad de Helsinki. Además, creó **Git**, sistema de control de versiones que hoy es fundamental para el desarrollo colaborativo de software. Su trabajo consolidó a Linux como la base de servidores, supercomputadoras y dispositivos móviles [15, 45].

Steve Jobs

Steve Jobs (San Francisco, Estados Unidos; 24 de febrero de 1955 – Palo Alto, 5 de octubre de 2011) fue un informático, empresario y diseñador industrial estadounidense, cofundador de **Apple Inc.**. Impulsó la creación de computadoras personales como el Apple II y el Macintosh, y posteriormente de dispositivos revolucionarios como el iPod, iPhone y iPad [4]. También fundó **NeXT Computer**, cuya tecnología influyó en macOS, y **Pixar**, pionera en animación digital. Su legado se ve magnificado por sus contribuciones a productos icónicos que han cambiado la forma en que vivimos y trabajamos. (Ver Figura I.11).

Bill Gates (William Henry Gates III)

Bill Gates Nació en Seattle, USA, en octubre de 1955. Es un programador, cofundador de la **Corporación Microsoft**, cuyo éxito se basó en el crecimiento de computadoras personales. Entre sus mayores logros está el lanzamiento de **Windows 95**, en cuya interfaz se destacan menús desplegables, botones de opción, de verificación, entre otros. Así mismo, el conjunto de programas de ofimática **Microsoft Office** en el que incluye Word, Excel, Power Point y Vision, que siguen siendo los más utilizados por los usuarios en el mundo. Además, en los últimos años se ha destacado por su labor filantrópica a través de la **Fundación Bill & Melinda Gates** [14, 40]. (Ver Figura I.12).

Figura I.11: *Efigie de Steve Jobs.* ***Fuente****: www.pixabay.com*

Figura I.12: *Efigie de Bill Gates.* ***Fuente****: www.pixabay.com*

Sitios de interés

(I) Explicación detallada sobre la evolución de los sistemas operativos: La evolución de los Sistemas Operativos: Un viaje desde MacOS hasta Android, Windows, iOS y Linux

(II) Explicación de la Fundación de Software Libre:

Free Software Foundation (FSF): que es una organización sin fines de lucro con la misión mundial de promover la libertad de los usuarios de computadoras.

(III) Explicación del Proyecto GNU.

Proyecto GNU. GNU acrónimo que significa "GNU'S Not Unix". El Proyecto GNU se fundó en 1984 para desarrollar un sistema operativo libre y compatible con Unix, que se llamaría GNU.

(IV) Interesante información que todo informático debe conocer: ¡Los 7 Programadores más relevantes de la Historia! . El legado de los pioneros de Unix, FSF, GNU, GNU/Linux, Lenguaje C, Facebook, en un entretenido post, personajes que dejaron un invaluable aporte para la revolución tecnológica en las ciencias de la computación.

Conclusiones y Lecciones aprendidas

Este capítulo ha sido enfocado en comprender los fundamentos básicos de los sistemas operativos. La literatura muestra que cada componente de su arquitectura desempeña un rol importante en la administración de los recursos computacionales. Se han señalado las principales funciones en las que se destaca la gestión de recursos, conectividad, seguridad, entre otros. Se han identificado varios hitos en la evolución histórica realizando una línea de tiempo imaginaria. Se han clasificados los sistemas operativos en libres y propietarios. Se ha analizado la filosofía del software libre que es un movimiento que defiende los derechos de los usuarios y se lo ha ejemplificado con el sistema operativo GNU/Linux.

Lecciones aprendidas

Dennis Ritchie fue el creador del lenguaje de programación C en 1972. Junto con Ken Thompson y Brian Kernighan, en los laboratorios de Bell desarrollaron el sistema operativo Unix.
La fundación de software libre (FSF) fue creada por Richard Mathew Stallman y otros investigadores en octubre de 1985. Este es un movimiento que defiende las libertades de los usuarios para estudiar, copiar, modificar y distribuir el software.
El kernel de Linux fue creado en 1991. En 1992 se creó el proyecto GNU/-Linux.
La primera versión de Microsoft Windows fue liberada en 1985. La versión actual es Windows 11 desde octubre de 2021.

Autoevaluación del Capítulo

1. **¿Qué sistemas operativos se ocupan comúnmente en los equipos de escritorio?**
   - Microsoft Windows, macOS.
   - Ubuntu server, Unix System V.
   - Fedora, Centos, Android, iOS.
   - Debian, Microsoft Windows server 2022, AIX .
2. **¿Qué opción define mejor qué es un sistema operativo?**
   - Un conjunto de programas que gestionan la seguridad y la conectividad.
   - Un conjunto de programas que gestionan la sincronización de entrada y salida.
   - Un conjunto de programas que controla el hardware y actúa como intermediario entre el usuario y la computadora.
   - Un conjunto de programas para gestión de usuarios.
3. **¿Cuáles son las funciones básicas de un sistema operativo?**
   - Seguridad.
   - Conectividad

- Actualizar la base de datos y el contenido en páginas Web.
- Administración de RAM, CPU e I/O Devices.

4. **¿A quién se le atribuye la creación del Sistema Operativo UNIX?**
   - Linus Torvalds.
   - Mark Zuckerberg.
   - Dennis Ritchie junto con Ken Thompson.
   - Richard Matthew Stallman
5. **¿En que año se liberó Linux?**
   - 1985.
   - 1992.
   - 1991.
   - 1972
6. **¿A quién se le atribuye la Fundación de Software Libre?**
   - Linus Torvalds.
   - Bruce Perens.
   - Richard Mathew Stallman.
   - Steve Jobs
7. **En la arquitectura de Linux ¿Cuáles son las funciones de la Shell?**
   - Es una arquitectura de datos en el que se incluye el árbol de directorios.
   - Gestión del árbol de directorios de Linux.
   - CLI, es una interfaz de línea de comandos.
   - Tiene como una función automatizar procesos en Linux mediante un lenguaje de programación intérprete.
   - Dentro de sus funciones está el analizar el tráfico en la red.
8. **¿Cuál es el símbolo de Linux?**
   - Gnu.
   - Apple.

- Tux.
- Windows.

9. **¿Qué tipo de sistema operativo es Windows NT?**
   - De equipo de escritorio.
   - De dispositivo móvil.
   - De Servidor.
   - De la computación en la nube
10. **¿Cuáles son las libertades de la filosofía del movimiento de software libre?**
    - Libertad de uso.
    - Libertad de estudio.
    - Libertad de distribución.
    - Libertad de modificación

CAPÍTULO II

Tecnologías de Virtualización y su evolución

Introducción

La virtualización es una tecnología disruptiva que transformó el funcionamiento de los centros de procesamiento de datos [6]. Sus primeros desarrollos se remontan a la década de 1960; no obstante, su verdadero auge y masificación se produjo en la década del 2000. En esencia, la virtualización constituye una estrategia de compartición de hardware que permite dividir un único equipo anfitrión en múltiples máquinas virtuales, cada una capaz de ejecutar diferentes sistemas operativos de manera independiente.

Esta tecnología posee diversos beneficios para la optimización del rendimiento de los recursos de hardware, lo que la convierte en una herramienta para la gestión de infraestructura tecnológica. Los sistemas operativos, son lo que se encargan de gestionar y coordinar las máquinas virtuales.

Entre las principales plataformas de virtualización utilizadas en la industria se encuentran **VMware**, **VirtualBox**, **Citrix Xen**, **Microsoft Hyper-V**, **VMware vSphere/ESXi**, **Virtual PC** y **KVM (Kernel-based Virtual Machine)**, cada una con características que serán analizadas en este capítulo.

Las tecnologías de la Virtualización han sufrido su evolución. Dada sus beneficios, se desarrolló un modelo de negocios basado en la nube computacional en la cual se han virtualizado los servidores para proveer diferentes servicios. Posteriormente, han surgido los contenedores, los cuales virtualizan el sistema operativo, pero no el hardware. En otras palabras, todas las aplicaciones comparten el kernel del equipo anfitrión.

Este capítulo contribuirá a que el lector comprenda los fundamentos de la virtualización y su importancia en el contexto de los sistemas operativos. Asimismo, le permitirá reconocer sus principales tecnologías y plataformas, aprenderá a configurarlas y entender cómo pueden convertirse en herramientas de entrenamiento y prueba para fortalecer su aprendizaje práctico. Además, el lector aprenderá sobre las capas de la nube computacional y sus modelos de servicios en las empresas. En definitiva, comprenderá que la nube computacional no reemplaza a la Virtualización. Sin embargo, la utiliza para hacer más eficiente sus servicios y aplicaciones. Finalmente, entenderá que los contenedores aparecen como una evolución de la virtualización en cuanto a eficiencia y portabilidad, especialmente para aplicaciones nativas de la nube computacional.

¡Bienvenidos!

Tecnologías de Virtualización

De acuerdo con un estudio preliminar en Fuertes et al., [49], las tecnologías de virtualización permiten la creación de máquinas virtuales y de entornos virtuales de red que emulan la funcionalidad de hardware, redes y servicios. La virtualización segmenta los recursos físicos de la computadora, como CPU, memoria interna, memoria de almacenamiento y dispositivos de entrada y salida [50]. Eso significa que los recursos del equipo anfitrión son divididos y asignados para cada máquina virtual que se pone en ejecución a nivel de software. Esto permite una mayor eficiencia en la utilización de los recursos de hardware, facilita la administración y despliegue de sistemas y aplicaciones. De esto se encarga el sistema operativo del equipo anfitrión. La Figura II.1 muestra una abstracción de a virtualización o consolidación de servidores, los cuales pueden inclusive formar parte una nube computacional.

En la década de los 2000, la virtualización transformó el funcionamiento de los centros de procesamiento de datos y la administración de la infraestructura de TI por los siguientes beneficios:

- Virtualización de servidores que implica dividir el servidor en múltiples máquinas virtuales autónomas, con su propio sistema operativo, direccionamiento físico, lógico y sus aplicaciones que permite aprovechar su verdadero potencial.
- Reducción de costos de inversión de hardware.
- Reducción de costos en la administración, electricidad, y mantenimiento de hardware.
- Reducción de costos de experimentación al ser un entorno controlado.
- Entorno de pruebas de software, bases de datos.
- Entorno virtual de red, seguro y controlado para la simulación y experimentación de ataques, pruebas de penetración, asimismo el análisis de vulnerabilidades.
- Entorno efectivo para la implementación de soluciones de recuperación ante desastres.
- Despliegue ágil de aplicaciones.
- Facilidad de configuración de servicios en red.

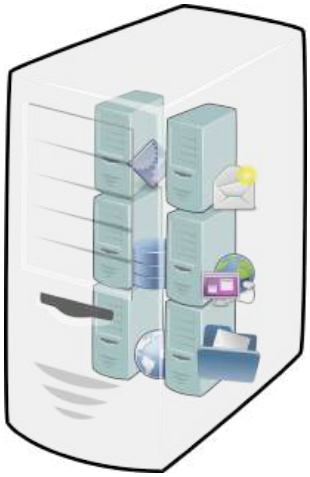


Figura II.1: Abstracción del funcionamiento de la virtualización de un servidor de hardware. **Fuente**: www.pixabay.com

A continuación, se exploran los distintos tipos de tecnologías de virtualización y plataformas comunes en la industria.

Virtualización Completa

La virtualización completa permite reproducir el funcionamiento total del sistema operativo del computador anfitrión en el sistema operativo hospedado, sin realizar modificaciones en el sistema operativo que lo soporta. Es una tecnología que permite la creación de múltiples máquinas virtuales, las cuales hospedan un sistema operativo, sobre un único hardware físico.

Un ejemplo tradicional de esta tecnología es VirtualBox de Oracle. De acuerdo con su sitio Web Virtual Box, Virtualbox, es una plataforma de virtualización de código abierto para uso personal y empresarial, bajo los términos de la Licencia Pública General GNU (GPL) versión 3.

VirtualBox es un software de virtualización de tecnología completa y de propósito general para para procesadores x86, AMD64 e Intel64 compatible con diferentes sistemas operativos, diseñado para equipos portátiles, ordenadores de sobremesa, servidores y sistemas integrados. Su simplicidad permite que se gestionen (crear, eliminar, configurar, clonar, etc.) varias máquinas virtuales con distintos sistemas operativos hospedados, que le hace versátil para poner en operación cualquier actividad técnica que demande recursos computacionales [49].

Para Virtualización

La para virtualización es una tecnología que utiliza un hipervisor para ejecutar múltiples máquinas virtuales con diferentes sistemas operativos en una solo equipo anfitrión o físico. Está tecnología, modifica el sistema operativo del host anfitrión, por tanto, no crea una capa adicional de virtualización sobre el hardware, lo que reduce el overhead (penalización o sobre carga), los costos de consumo energético, y mejora la eficiencia de uso de los recursos [33]. El hipervisor gestiona el hardware, asignando CPU, memoria interna, almacenamiento, dispositivos de I/O y de netwotking según las necesidades que tenga cada máquina virtual (i.e., cada VM es independiente y aislada de otras), lo que impide que se colapse todo el sistema virtualizado [49].

Entre las principales plataformas de virtualización por hipervisor se pueden citar VMware ESXi, Xen, Microsoft Hyper-V. La Figura II.2 muestra una abstracción de sus tecnologías y plataformas más utilizadas en el industria.

Figura II.2: Tecnologías de virtualización y sus plataformas más utilizadas en la industria. **Fuente**: Esta ilustración fue creada por los autores utilizando las herramientas Google Gemini IA Generativa

Microsoft Hyper-V:

De acuerdo con su sitio web oficial, Microsoft Hyper-V, es una plataforma de virtualización de Microsoft basada en el hipervisor de Windows, que permite a los usuarios crear y administrar varias máquinas virtuales. Proporciona una solución escalable para ejecutar varios sistemas operativos en cada maquina virtual, en un solo servidor físico que operan aisladamente. Hyper-V

permite la consolidación de servidores, la eficiencia en la asignación de recursos y la posibilidad de recuperación ante desastres.

A nivel del sistema operativo

Permite que múltiples entornos de usuarios aislados y contenedores se ejecuten en un único sistema operativo hospedado. Los contenedores son imágenes independientes, portátiles y ejecutables que contienen aplicaciones de software y sus dependencias. Se utilizan para implementar y ejecutar aplicaciones en diferentes entornos, como desarrollo, pruebas y producción.

Hipervisores: Tipos y Funciones

El hipervisor constituye el componente central en la virtualización basada en máquinas virtuales, ya que actúa como intermediario entre el hardware físico y los sistemas operativos invitados. Su función principal consiste en administrar y asignar los recursos físicos del sistema, tales como CPU, memoria, almacenamiento y dispositivos de entrada/salida, a cada máquina virtual, garantizando aislamiento, estabilidad y eficiencia operativa [127].

El hipervisor es el responsable de crear una separación del hardware que permita a los sistemas operativos hospedados ejecutarse como si estuvieran operando sobre una máquina física. Esto es fundamental para que las máquinas virtuales se encuentren aisladas entre sí, evitando interferencias y fallos en cascadas que comprometan la estabilidad del sistema anfitrión [121].

Existen dos tipos principales de hipervisores ampliamente reconocidos en la literatura. El **hipervisor de tipo 1**, también denominado *bare-metal*, se ejecuta directamente sobre el hardware físico sin la intervención de un sistema operativo anfitrión. Este enfoque reduce la sobrecarga introducida por capas adicionales de software, mejora el rendimiento general y refuerza la seguridad del sistema, razón por la cual es ampliamente utilizado en centros de datos y entornos empresariales. Ejemplos representativos de este tipo de hipervisor incluyen VMware ESXi, Microsoft Hyper-V Server y Xen [11].

Por otro lado, el **hipervisor de tipo 2** se ejecuta sobre un sistema operativo anfitrión, lo que lo hace más accesible para entornos de escritorio, laboratorios académicos y escenarios de prueba. Aunque este enfoque introduce una capa adicional de abstracción que puede afectar ligeramente el rendimiento, sigue siendo una opción válida para fines educativos, desarrollo de software y experimentación. VirtualBox y VMware Workstation son ejemplos comunes de

hipervisores de tipo 2 ampliamente utilizados [127].

La selección del tipo de hipervisor depende del uso al cual se va a exponer, los requisitos de rendimiento, la criticidad de los sistemas virtualizados y la infraestructura disponible. En entornos de producción, los de tipo 1 son los más recurrentes, mientras que los de tipo 2 resultan útiles para el aprendizaje, pruebas y desarrollo.

Gestión de Recursos en Entornos Virtualizados

La gestión eficiente de recursos constituye uno de los pilares fundamentales de la virtualización moderna. Los sistemas virtualizados permiten asignar dinámicamente recursos como CPU, memoria y almacenamiento a las máquinas virtuales en función de su carga de trabajo, lo que optimiza el uso del hardware disponible y reduce el desperdicio de recursos físicos [121].

Los hipervisores modernos se encuentran constituidos por mecanismos avanzados de gestión (sobreasignación de memoria, la planificación de CPU y el balanceo dinámico de carga entre máquinas virtuales). Donde, permiten que múltiples máquinas virtuales compartan recursos sin afectar el rendimiento global del sistema, siempre y cuando se respeten las políticas de asignación establecidas [117].

Asimismo, se emplean mecanismos de control y limitación de recursos, tales como cuotas, reservas y afinidad de procesadores, con el objetivo de garantizar que las aplicaciones críticas mantengan niveles adecuados de desempeño incluso en escenarios de alta demanda. Estas capacidades resultan esenciales en infraestructuras empresariales, donde la disponibilidad y la previsibilidad del rendimiento son factores clave [127].

Desde una perspectiva administrativa, la gestión de recursos en entornos virtualizados facilita la planificación de capacidad, el monitoreo del rendimiento y la automatización de tareas operativas. Estas características han convertido a la virtualización en un componente indispensable de las infraestructuras modernas y de los entornos de computación en la nube. La Figura II.3, muestra un escenario virtual de red, con varias máquinas virtuales en ejecución.

El autor en sus funciones de docencia e investigación desde el 2010 viene empleando entornos de red virtuales controlados para el proceso de enseñanza aprendizaje práctico de redes de computadores, sistemas operativos, seguridad de la información y ciberseguridad.

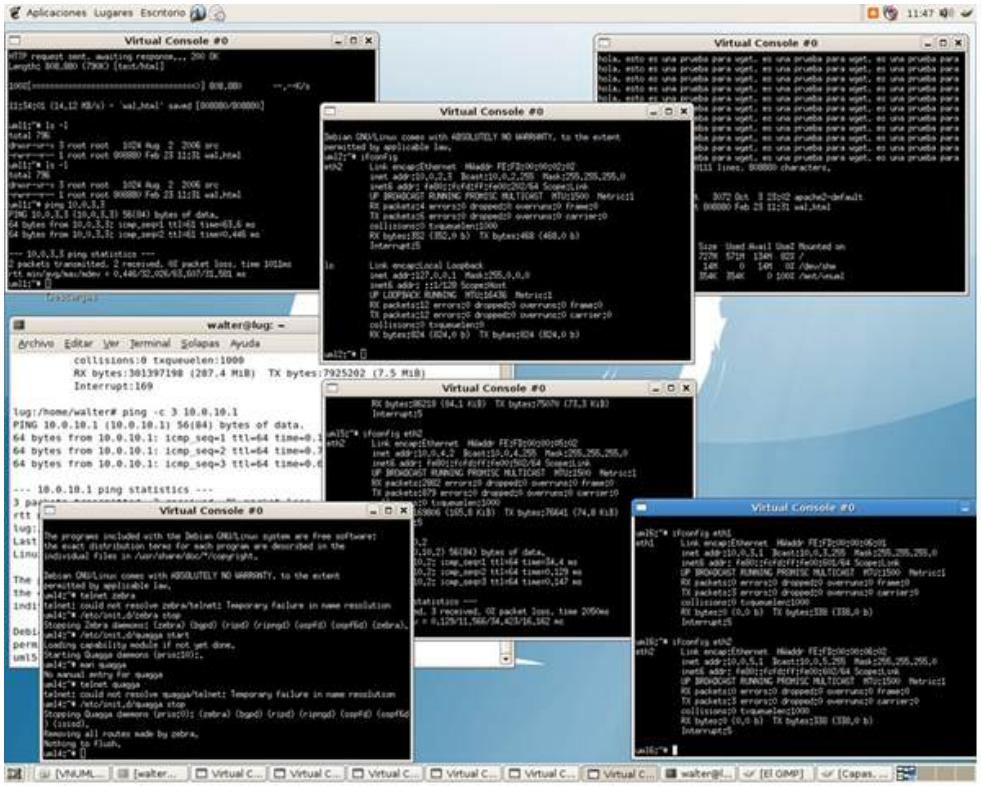


Figura II.3: Escenario virtual de red controlado

.

La Figura II.3 muestra un escenario virtual controlado con seis máquinas virtuales configuradas mediante la plataforma VMware Player. Como se puede observar, en un solo host anfitrión, se levantó una plataforma experimental con varias máquinas virtuales, cada una de ellas con Ubuntu desktop en la que se instaló la base mínima del sistema operativo para su funcionamiento. A través de estos entornos se pueden desarrollar diferentes experimentos, con lo cual se disminuyen los costos de inversión de hardware, de software, mantenimiento y energización [49], [50].

Como se mencionó en la Introducción de este capítulo, las tecnologías de virtualización, como tecnologías disruptivas, están en constante evolución. Dados sus beneficios, la industria desarrolló modelos de negocio basados en la nube mediante la virtualización de servidores para ofrecer una gama de servicios, aplicaciones, infraestructura y plataformas de desarrollo. Posteriormente, surgieron los contenedores, que virtualizan el sistema operativo, pero no el hardware, de modo que todas las aplicaciones comparten el kernel del equipo host [102]. Esta evolución se alinea fácilmente con el desarrollo de software empresarial, como se muestra en la Figura II.4. A continuación, se explicará dicha evolución.

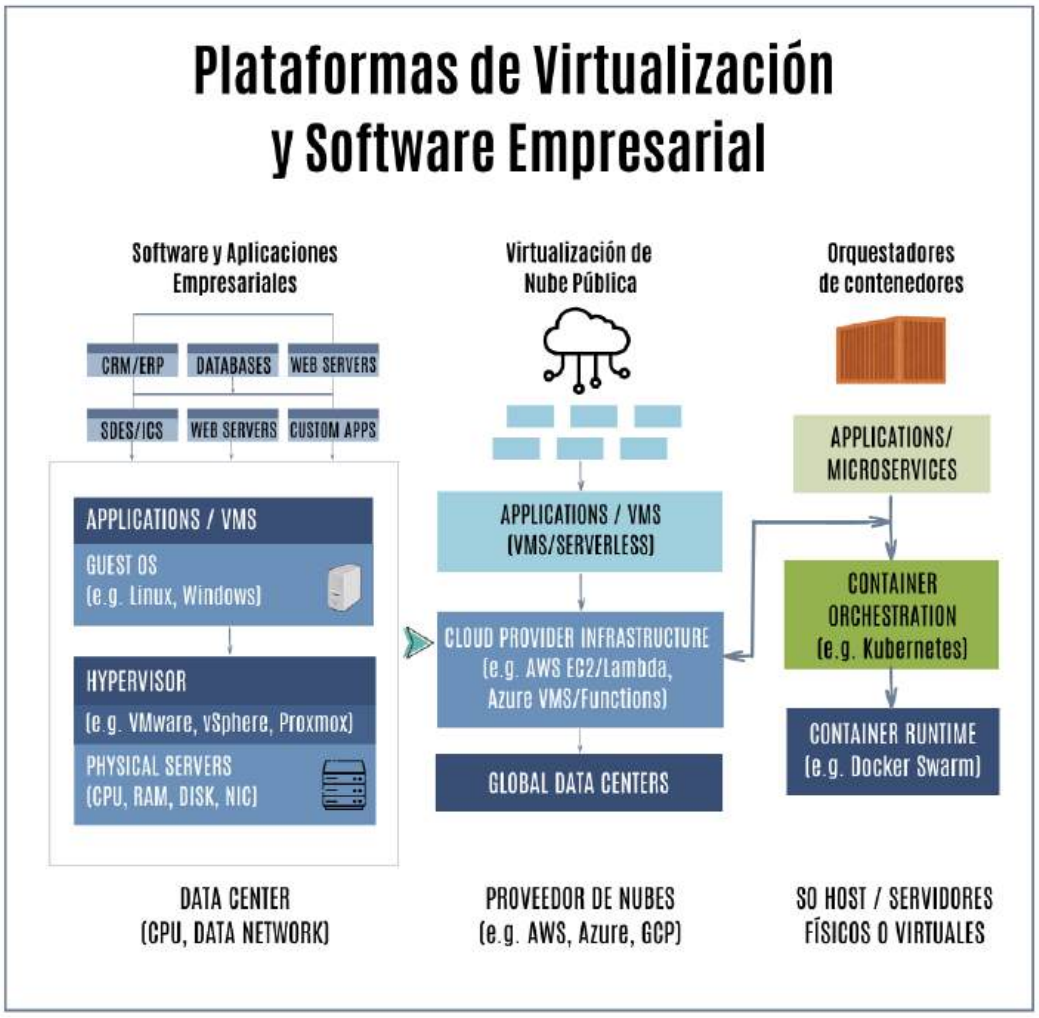


Figura II.4: Evolución de la Virtualización y su relación con el software empresarial. **Fuente:** Esta ilustración fue creada por los autores utilizando Google Gemini IA.

Virtualización y Computación en la Nube

La virtualización constituye la base tecnológica sobre la cual se construyen los modelos modernos de computación en la nube. Gracias a la capacidad de abstraer el hardware físico y crear múltiples entornos virtuales aislados, es posible ofrecer recursos informáticos como servicios bajo demanda a través de Internet, permitiendo un uso flexible y eficiente de la infraestructura [85].

Los principales modelos de despliegue de computación en la nube son: La nube pública, que es compartida por múltiples organizaciones y usuarios y

gestionada por proveedores externos. La nube privada, exclusiva de una sola organización y caracterizada por sus amplios sistemas de control y seguridad. La nube híbrida, que combina las ventajas de las nubes pública y privada, compartiendo datos y ofreciendo mayor flexibilidad. En los últimos años, la nube comunitaria ha surgido como un recurso compartido para organizaciones específicas con intereses comunes. La Figura II.5 es una representación visual de sus modelos de despliegue:

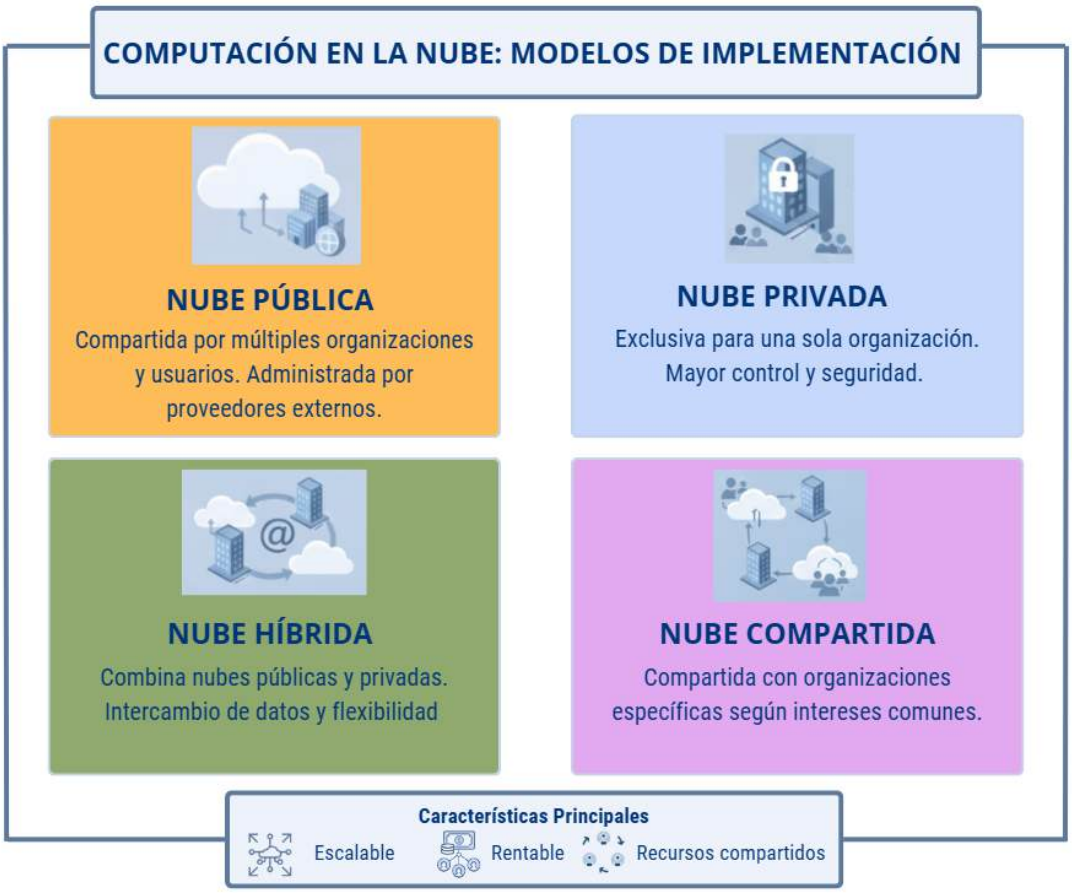


Figura II.5: Modelos de despliegue de la Computación en la Nube. **Fuente:** Esta ilustración fue creada por los autores utilizando Copilot IA

.

Los modelos de servicio en la computación en la nube incluyen Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS), Software como Servicio (SaaS) y Funciones como servicio (FaaS). En el modelo IaaS, la virtualización permite a los proveedores ofrecer máquinas virtuales, almacenamiento y redes configurables; en PaaS, facilita entornos de desarrollo estandarizados; en SaaS, posibilita la entrega de aplicaciones completas al usuario final sin necesidad de gestionar la infraestructura subyacente [18]. Finalmente, la capa

Faas permite ejecutar código en la nube sin necesidad de administrar servidores, sistemas operativos o infraestructura. La Figura II.6 es una representación visual de sus modelos de servicio:

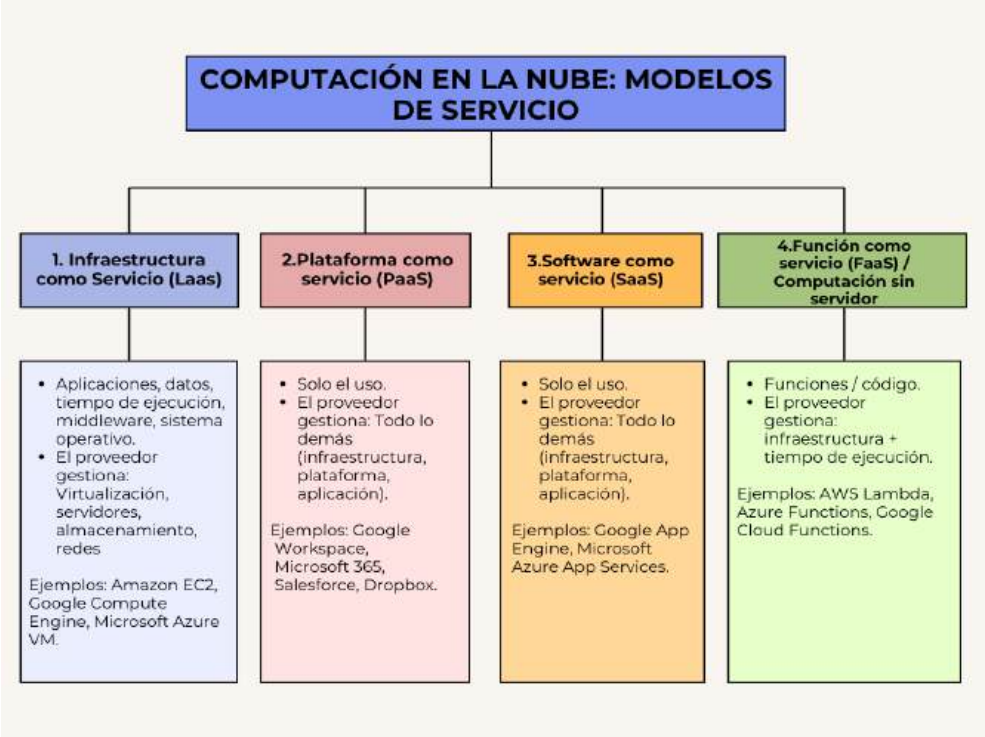


Figura II.6: Modelos de servicio de la Computación en la Nube. **Fuente:** Esta ilustración fue creada por los autores utilizando Google Gemini IA

.

En este contexto, tanto las máquinas virtuales como los contenedores desempeñan un papel fundamental. Las máquinas virtuales proporcionan un alto grado de aislamiento y compatibilidad entre diferentes sistemas operativos, mientras que los contenedores ofrecen ligereza, portabilidad y rapidez en el despliegue de aplicaciones. La combinación de ambas tecnologías permite a los proveedores de servicios en la nube equilibrar seguridad, rendimiento y escalabilidad [116].

Asimismo, la virtualización facilita la migración en caliente de máquinas virtuales, la alta disponibilidad y la recuperación ante desastres, por lo cual es esencial en entornos de nube pública, privada e híbrida. Debido a esto, comprender sobre la virtualización resulta útil para el estudio y diseño de arquitecturas de computación en la nube.

Proveedores de la nube

Son las compañías que administran y operan los centros de datos físicos y las plataformas de software que ofrecen recursos informáticos como servicio en la nube computacional a través de Internet. Entre ellos, se pueden distinguir:

Hiperescaladores: Que son los gigantes globales que ofrecen servicios integrales a gran escala y dominan el mercado. Por ejemplo, Amazon Web Service (AWS), Microsoft Azure, Google Cloud Platform (GCP) [32].

Proveedores especializados: Se enfocan en nichos, regiones o tecnologías específicos, como por ejemplo Coreweave & Cloudfleet. Otros especializados en infraestructuras aceleradas por GPU para Inteligencia Artificial, Aprendizaje Automático y renderizado; Otros como DigitalOcean & Linode (Akamai), enfocados en simplicidad y rendimiento para desarrolladores y PyMEs. Finamente, Rackspace para servicios gestionados multinube y Salesforce que lidera en CRMs y aplicaciones SaaS empresariales [54].

Ecosistema de la nube

Se refiere a la red interconectada de servicios, entidades, soluciones, usuarios, tecnologías y proveedores de soporte de computación en la nube que interactúan con la nube, pues permiten la transmisión, recepción, gestión y consumo de recursos y servicios de la nube. Algunos de sus principales elementos son:

Proveedores de soporte: Que son los proveedores de software independientes que crean aplicaciones que se ejecutan en la infraestructura de la nube computacional.

Socios de servicio: Que son los proveedores de servicios gestionados y consultores que ayudan a las organizaciones a migrar, implementar y gestionar sus entornos de nube computacional.

Usuarios: Que son los desarrolladores y usuarios finales que consumen los servicios y crean soluciones.

En resumen, los proveedores ofrecen la plataforma y los recursos, mientras que el ecosistema incluye todos los socios, el software y los usuarios que dependen de la funcionalidad de la nube computacional. La Figura II.7 muestra los principales proveedores de Cloud en la industria:

Figura II.7: Principales proveedores de servicios en la Nube. **Fuente**: Esta ilustración fue creada por los autores utilizando la herramienta Copilot IA Gen

Ejercicios propuestos

Ejercicio 1

¿Cuál es la diferencia entre una tecnología y una plataforma de virtualización?

Ejercicio 2

¿Qué ventajas posee la tecnología de virtualización completa y para virtualización?

Ejercicio 3

¿Qué función cumple el hipervisor?

Ejercicio 4

¿Cuáles son los modelos de servicio de la nube computacional?

Ejercicio 5

¿Cuáles son los modelos de despliegue en Cloud?

Ejercicio 6

Investigue cómo han evolucionado las tecnologías de la virtualización.

Ejercicio 7

¿Cuáles son las técnicas de contenedores más utilizados en la industria?

Ejercicio 8

¿En qué consiste la capa FaaS?

Ejercicio 9

Investigue un caso real de una empresa que utilice microservicios y describa los beneficios obtenidos con esta arquitectura.

Ejercicio 10

¿Cuáles son los mayores proveedores de servicios de nube computacional en la industria?

Conclusiones del Capítulo

Este capítulo ha proporcionado una introducción conceptual de la virtualización, que son gestionadas por los sistemas operativos. La Virtualización es una tecnología predominante en la eficiencia de los sistemas computacionales. Los lectores han aprendido en qué consiste la virtualización, su tipología, sus plataformas y cómo ha evolucionado atravesando por la computación en la nube y los contenedores. La virtualización es un componente esencial de la infraestructura de TI moderna, que permite una mayor eficiencia, escalabilidad y reducción de costos mediante plataformas como VirtualBox, VMware, Hyper-V, KVM, Citrix, entre otras. Se han explorado conceptos esenciales de máquinas virtuales, hipervisor, nube computacional y contenedores. Además, mediante la plataforma Virtual Box, se ha comprendido que se trata de una plataforma de código abierto que permite crear, eliminar, configurar y clonar

máquinas virtuales. Mediante Virtual Box se ha implementado un entorno virtual de red seguro y controlado. Además, en este capítulo han sido explorados los modelos de servicios en la nube computacional como una evolución de la virtualización (IaaS, PaaS, SaaS, FaaS), los cuáles ofrecen distintos niveles de control y responsabilidad. Esto posibilita a las organizaciones adaptar las soluciones a sus necesidades de desarrollo, gestión y administración de clientes, proveedores e interesados (stakeholders). Por otro lado, se identificaron algunos modelos de implementación de la nube computacional en nube pública, privada, híbrida y comunitaria. Estos modelos deben alinearse a la gobernanza de datos, sus políticas de seguridad y los objetivos estratégicos de la organización. Así mismo, se identificaron entre los principales proveedores de nube a AWS, Azure y Google Cloud, que conforman un ecosistema competitivo que ofrece servicios especializados que impulsan la innovación, la IA, la analítica de datos y la integración continua en toda la empresa. Los ejemplos prácticos y problemas resueltos proporcionados a lo largo del capítulo han permitido consolidar estos conocimientos, ayudando al lector a desarrollar habilidades para configurar entornos virtuales de red. Al finalizar este capítulo, el lector no solo comprende los fundamentos de las tecnologías de virtualización, sino que además aprende a caracterizarlas y a distinguir una nube computacional y los contenedores.

Lecciones aprendidas

- La virtualización es una tecnología disruptiva que transformó el funcionamiento de los centros de procesamiento de datos. Sus primeros desarrollos se remontan a la década de 1960; no obstante, su verdadero auge y masificación se produjo en la década de 2000.
- La virtualización constituye una estrategia de compartición de hardware que permite dividir un único equipo anfitrión en múltiples máquinas virtuales, cada una capaz de ejecutar diferentes sistemas operativos de manera independiente.
- La virtualización es una tecnología que utiliza un hipervisor para ejecutar múltiples máquinas virtuales con diferentes sistemas operativos en una solo equipo anfitrión o físico.
- La nube computacional, su tipología, capas y sus modelos de servicios en las empresas son una evolución de la Virtualización que no la reemplaza, sino más bien, que la utiliza para hacer más eficiente sus servicios y aplicaciones.
- Los contenedores también son una evolución de la virtualización que favorecen la eficiencia y portabilidad, especialmente para aplicaciones nativas de la nube computacional.
- Los modelos de servicios en la nube computacional son una evolución de la virtualización (IaaS, PaaS, SaaS, FaaS), los cuáles ofrecen distintos niveles de control y responsabilidad.

Sitios de interés

La Guía de cloud computing: Principios y práctica. Springer. https://doi.org/10.1007/978-3-319-96812-4

Google Cloud security whitepapers. Retrieved from https://cloud.google.com/security

Amazon Web Services. (2023). Netflix Case Study. Retrieved from https://aws.amazon.com/solutions/case-studies/netflix/

VMware. (2024). Understanding Virtualization. Retrieved from https://www.vmware.com

Amazon Web Services (AWS). (2024). Overview of Amazon Web Services. Retrieved from https://aws.amazon.com/whitepapers.

Virtual Box home page, URL: https://https://www.virtualbox.org/

Autoevaluación del Capítulo

Responda las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿Cuál es el acrónimo de las Siglas FaaS en cloud computing?**
   - Functioning as a Service.
   - Function as a Service
   - Fan as a Service.
   - Furniture as a Service.
2. **Defina Virtualización**
   - Es una tecnología antigua que ha quedado en obsolescencia.
   - Es una tecnología disruptiva que divide lógicamente a un equipo anfitrión en diferentes máquinas virtuales cada una con diferente sistema operativo
   - La virtualización en esencia es una tecnología de inversión de hardware.
   - La virtualización divide físicamente a un equipo anfitrión en varias máquinas virtuales.
3. **¿Cuál es la función del hipervisor?**
   - Es un artefacto de software que agrupa recursos computacionales, como el CPU, la memoria principal y el almacenamiento, y los asigna entre las máquinas virtuales.
   - Prioriza los procesos con menor memoria.
   - Asigna mayor cantidad de memoria al servidor virtualizado.
   - Alterna memoria entre las máquinas virtuales sin seguir un orden específico.

4. **¿Cuál es la diferencia entre tecnología de virtualización y plataforma?**
   - La tecnología de virtualización se encarga de articularse con la nube computacional.
   - La plataforma de virtualización es aquella que permite gestionar las máquinas virtuales. La tecnología de virtualización en cambio es la responsable de gestionar el hardware subyacente
   - La plataforma de virtualización en todos los casos es de código abierto.
   - La mejor tecnología de virtualización es aquella que en su ejecución incremente el overhead que es una penalización al procesador por atener a la capa de virtualización.
5. **¿En qué se diferencia la nube con los contenedores?**
   - La nube computacional es un modelo de servicios dependiente de los gobiernos de turno.
   - La nube no tiene dependencia con el servicio de Internet como es el caso de los contenedores.
   - La nube es un modelo de servicio utilizando el Internet que incorpora servidores virtualizados, mientras que los contenedores asilan las aplicaciones sin necesidad de sistemas operativos virtualizados.
   - La nube es un espacio inseguro que tiene alta dependencia con el Internet, mientras que los contenedores no.
6. **¿En qué consiste VirtualBox?**
   - Es una plataforma de virtualización comercial para uso personal y empresarial.
   - Software de virtualización completa y de propósito general para hardware x86_64, compatible con diferentes sistemas operativos, diseñado para equipos portátiles, ordenadores de sobremesa, servidores y sistemas integrados.
   - Se trata de una plataforma de uso comercial y de propósito específico.
   - Se trata de una plataforma compleja, que le impide ser versátil para poner en operación cualquier actividad técnica que demande recursos computacionales.

7. **¿En qué consiste el modelo de Software como Servicio?**
   - Es un modelo de servicio que provee infraestructura de red y de base de datos para prestación de servicios en el Internet
   - Es un modelo de servicio que provee aplicaciones de software para hacer más eficiente los procesos de las empresas.
   - Es un modelo de servicios que provee la plataforma de desarrollo de software.
   - Es un modelo de servicio que permite a los usuarios ejecutar aplicaciones sin tener que administrar la infraestructura.
8. **Defina nube computacional pública**
   - Aquella que es compartida por múltiples organizaciones y usuarios y es administrada por terceros.
   - Es de exclusiva de una única organización.
   - Aquella que combina las características de privada y comunitaria.
   - Aquella que es utilizada por organizaciones específicas con interés comunes.
9. **Defina nube privada**
   - Aquella que combina las características de privada y comunitaria.
   - Aquella que es compartida por múltiples organizaciones y usuarios y es administrada por terceros.
   - Modelo de servicios que están destinados a un solo cliente
   - Aquella que es utilizada por organizaciones específicas con interés comunes.
10. **Los Dockers son contenedores que funcionan de la siguiente manera:**
    - Es la tecnología de contenedores que posibilita la creación y el uso de los contenedores de Windows.
    - Permite utilizar los contenedores cual fueran máquinas virtuales.
    - Utiliza el kernel de Linux y sus funciones para dividir procesos y ejecutarlos por separado para conservar la seguridad que se obtendría con los sistemas individuales.
    - Ninguna de las anteriores.

CAPÍTULO III

Clasificación de Sistemas Operativos

Introducción

En el ámbito de los sistemas operativos, la diversidad de arquitecturas y enfoques responde a las distintas necesidades y desafíos que enfrentan los dispositivos y entornos computacionales contemporáneos. Desde computadoras personales y servidores hasta sistemas embebidos y redes distribuidas, cada tipo de sistema requiere un enfoque específico para la gestión de recursos, la ejecución de aplicaciones y la interacción con el hardware. Este capítulo se centra en explorar las diferencias fundamentales entre varios tipos de arquitecturas de sistemas operativos, su relevancia y su implementación en diferentes entornos.

Los temas de este capítulo abarcan los diferentes tipos de sistemas operativos, desde las estructuras monolíticas hasta los sistemas distribuidos y embebidos, así como conceptos como microkernels y sistemas multitarea y multiprocesador. Estos temas son importantes para entender el funcionamiento de los dispositivos que se utilizan a diario, asimismo, cómo se desarrollan sistemas más avanzados, los cuales se adaptan para mejorar su rendimiento, seguridad y capacidad de crecimiento.

Al finalizar este capítulo, el lector será capaz de identificar las características principales de cada tipo de sistema operativo, entender los compromisos de diseño involucrados y apreciar las razones detrás de la elección de una arquitectura sobre otra en escenarios de la vida real.

Exploremos juntos la estructura y funcionalidad de los sistemas operativos.

¡Bienvenidos!

Clasificación de Sistemas Operativos

Sistemas Operativos Monolíticos

Los sistemas operativos monolíticos representan uno de los diseños más antiguos y tradicionales, donde el núcleo del sistema operativo (kernel) proporciona múltiples servicios desde un único bloque de software compacto y altamente interconectado. Este tipo de arquitectura es conocida por su eficiencia y rapidez, dado que todos los componentes fundamentales del sistema operativo, como la gestión de memoria, los dispositivos de entrada/salida, y el manejo de archivos, se ejecutan en el mismo espacio de memoria (véase Figura III.1). Un ejemplo clásico de un sistema operativo monolítico es Linux.

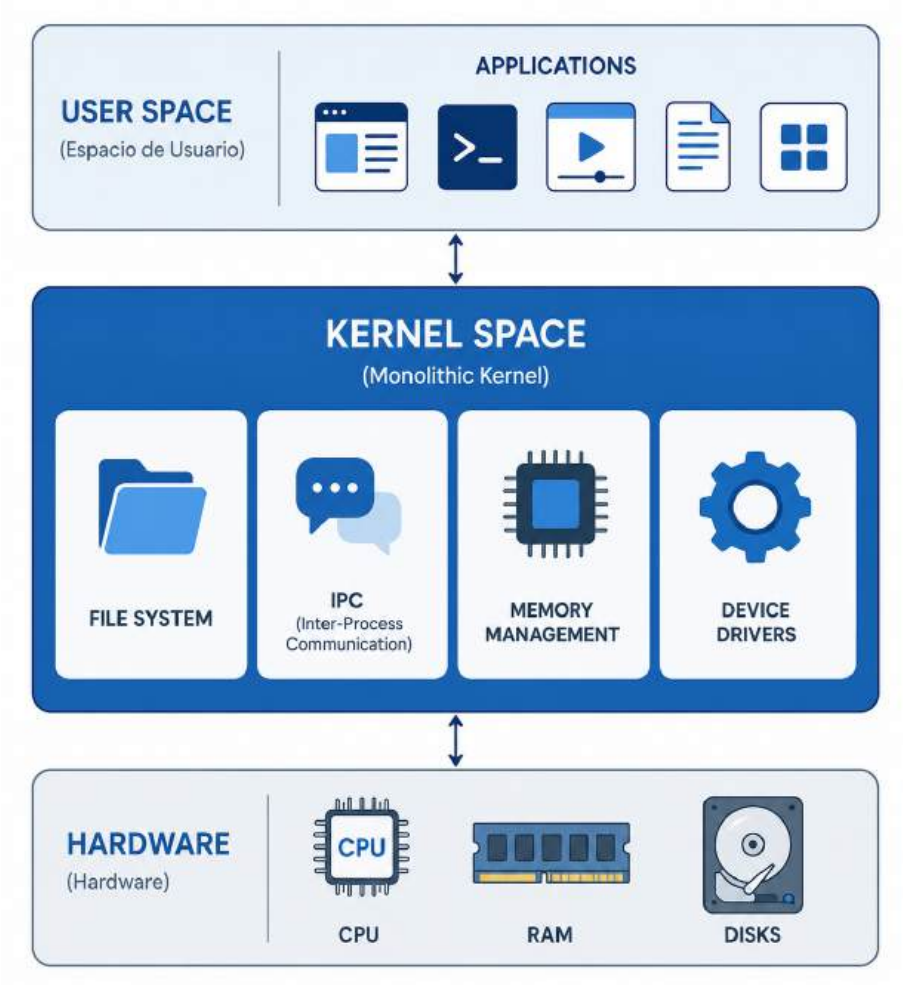


Figura III.1: Ejemplo visual de S.O. de arquitectura monolítica. **Fuente**: Ilustración creada por los autores utilizando Microsoft Copilot IA Generativa.

Sistemas Operativos con Microkernel

La principal diferencia contra los sistemas monolíticos, es que los que utilizan microkernel tienen una estructura modular, es decir, que el núcleo solo realiza las tareas más importantes, como la comunicación entre procesos y el manejo de la memoria. Otras tareas, como los drivers de dispositivos y los sistemas de archivos, se ejecutan como programas separados, lo que significa que el sistema es más seguro y menos propenso a fallos, debido a que si existe algún error, este no afecta a todo el sistema en general. Algunos ejemplos de este sistema operativo son Minix y QNX (véase Figura III.2).

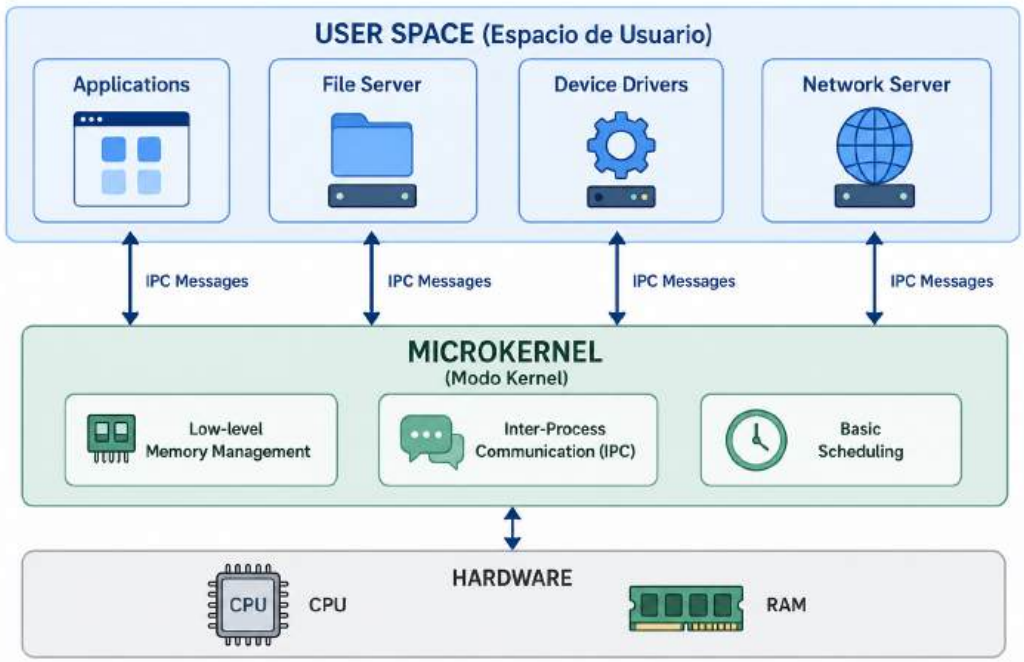


Figura III.2: Ejemplo visual de S.O. de arquitectura microkernel. **Fuente**: Ilustración creada por los autores utilizando ChatGPT IA Generativa.

Para la gestión de tareas, es fundamental conocer acerca de la multiprogramación y la multitarea, debido a que se sirve como ilustración de la administración y optimización de los recursos del sistema para la ejecución simultánea de múltiples programas y procesos.

Multiprogramación

Es una técnica antigua que permite que múltiples procesos puedan ser cargados en la memoria principal al mismo tiempo y sean ejecutados de manera concurrente (aparentemente) por el procesador o CPU [95]. Aunque se conoce

que un solo procesador puede atender a un solo proceso hasta que no se le despojen sus recursos, la idea de múltiples procesos se logra cuando cualquier proceso de entrada y salida es atendido al residir en la memoria compartida.

Multitarea

En este tipo de sistemas operativos, los usuarios pueden ejecutar varias tareas simultáneamente. Por ejemplo, mientras el usuario está escribiendo en Latex, a la vez, está escuchando música a través del YouTube, tiene WhatsApp Web, con lo cual recibe los mensajes en línea. Además, se encuentra entrenando un modelo con Google Colab.

Sistemas Operativos de multiprocesamiento

Son aquellos que tienen varios procesadores o varios núcleos (dualcore, quadcore, hexacore, octacore, y más) en el mismo equipo, lo cual mejora el desempeño del sistema en razón de que puede realizar varias tareas simultáneamente. Estos sistemas ejecutan procesos o hilos en paralelo, al distribuir la carga de trabajo entre varios procesadores, dando como resultado una mejora en el rendimiento y en la capacidad de respuesta del sistema.

Sistemas Operativos Distribuidos

Los sistemas operativos distribuidos gestionan un grupo de computadoras independientes y las hacen aparecer a los usuarios como un único sistema coherente. Esta configuración permite compartir recursos y procesar cargas de trabajo de manera distribuida, lo que puede mejorar significativamente el rendimiento y la disponibilidad del sistema (véase la Figura III.3).

Figura III.3: Ejemplo visual de un S.O. distribuido. **Fuente**: Ilustración creada por los autores utilizando ChatGPT IA Generativa.

Sistemas Operativos de Red

Los sistemas operativos de red facilitan la conexión de computadoras y el uso compartido de recursos dentro de una red. Estos sistemas operativos proporcionan funcionalidades esenciales, como el acceso a archivos remotos, la impresión en red y la comunicación entre procesos a través de la red (Véase la Figura III.4).

Sistemas Operativos Embebidos

Están diseñados para operar en dispositivos especializados que no son típicamente considerados computadoras, como routers, electrodomésticos y sistemas de control industrial. Estos sistemas se encuentran optimizados para ejecutar un conjunto específico de tareas con requisitos mínimos de recursos, donde se asegura una operación eficiente en entornos que poseen recursos de hardware limitados (Véase la Figura III.5).

Figura III.4: Representación visual de un S.O en Red para gestionar procesos desde múltiples estaciones de trabajo. **Fuente**: Ilustración creada por los autores utilizando Chat GPT IA Generativa.

Figura III.5: Ejemplo de un sistema operativo embebido en un smart watch. Estos dispositivos utilizan sistemas operativos especializados que están diseñados para gestionar recursos eficientemente en un espacio limitado. **Fuente**: www.pixabay.com

Ventajas Comparativas de las Arquitecturas de Sistemas Operativos

En este apartado, exploraremos las ventajas de diferentes arquitecturas de sistemas operativos comparándolas directamente, lo cual nos permitirá entender por qué se podría preferir una arquitectura sobre otra en distintos escenarios de uso.

Eficiencia y Rendimiento

- **Monolíticos**: Ofrecen un rendimiento óptimo y eficiencia debido a que todos los servicios importantes se ejecutan en el mismo espacio de memoria, reduciendo la latencia en la comunicación entre servicios.

- **Microkernel**: Presenta retos en cuanto a desempeño y complejidad computacional, sin embargo, favorece las aplicaciones en microservicios.

Seguridad y Estabilidad

- **Microkernel**: Mejora la seguridad del sistema puesto que dispone de privilegios de acceso tanto al hardware como a los servicios que se ejecutan en modo usuario. Además, gracias a su arquitectura modular, el aislamiento de los servicios mejora su tolerancia a fallos. Es decir, si un módulo falla, el resto del sistema seguirá en funcionamiento.

- **Monolíticos**: Al contrario de microkernel, en el caso de fallo de un servicio, eso podría afectar a todo el sistema, con lo cual, en este caso, disminuye la seguridad.

Escalabilidad y Flexibilidad

- **Sistemas Operativos Multiprocesador y Distribuidos**: Se caracterizan por el balanceo de carga, alta disponibilidad, transparencia, y procesan la información como si tratara de un único sistema.

- **Sistemas Operativos de Red**: Permiten una fácil expansión y administración de recursos compartidos en una red.

Costo-Efectividad y Administración de Recursos

- **Sistemas Operativos de Red**: Reducen costos al centralizar recursos que pueden ser compartidos por muchos usuarios, como impresoras y servidores de archivos.
- **Sistemas Operativos Embebidos**: Son implementados para ejecutar funciones dedicadas o específicas en tiempo real, potenciados con pocos recursos y para un bajo consumo de energía.

Sitios de interés

(I) Video sobre la estructura y eficiencia de sistemas operativos microkernel: Microkernel - Estructuras de Sistemas Operativos

(II) Guía sobre las ventajas de los microkernels sobre sistemas monolíticos: La arquitectura microkernel: futuro de los sistemas operativos

(III) Exploración de técnicas de multiprogramación y multitarea en sistemas operativos: Multiprogramación y Multitarea: Eficiencia y Simultaneidad en la Ejecución de Procesos

(IV) Introducción a sistemas operativos multiprocesador y sus beneficios en la era moderna: Sistemas Operativos Multiprocesador: Optimización de Tareas Paralelas

(V) Artículo sobre la gestión de recursos y tareas en sistemas operativos distribuidos y de red: Sistemas Operativos Distribuidos y de Red: Cooperación y Conectividad

(VI) Artículo sobre sistemas operativos embebidos y ejemplos de implementación en dispositivos: XEOS PARA EL DESARROLLO DE SOFTWARE BASE DE SISTEMAS EMBEBIDOS

Conclusiones y Lecciones aprendidas

Este capítulo ofreció una vista amplia y detallada sobre la variedad de sistemas operativos, destacando sus estructuras, funcionalidades y aplicaciones específicas. Desde sistemas operativos monolíticos y microkernels, hasta enfoques modernos como sistemas multiprocesador y distribuidos, cada tipo se examinó en función de sus necesidades tecnológicas específicas.

Lecciones aprendidas

- La diferencia entre un sistema monolítico y de microkernel está en como el espacio de usuario y del kernel están asignados; cada arquitectura plantea una estrategia distinta con sus ventajas y desventajas.
- La multiprogramación y la multitarea sirven para optimizar el desempeño del CPU y la memoria; permiten ejecutas múltiples procesos o tareas simultáneamente.
- Al sumar más procesadores, los sistemas multiprocesador consiguen dividir la carga de trabajo, lo que acelera la velocidad a la que se completan las tareas.
- Cuando los dispositivos no disponen de mucha potencia, los sistemas operativos embebidos, que están hechos a medida, son una solución viable.

Autoevaluación del Capítulo

1. **¿Cuáles son las ventajas de las distribuciones Ubuntu para escritorio y servidor?**
   - Forman parte de la filosofía del movimiento de software libre.
   - Tienen costo de licenciamiento.
   - Son de código abierto y de libre distribución.
   - Ninguna respuesta es correcta.
2. **¿Qué distingue a un sistema operativo de arquitectura microkernel?**
   - El colapso de un módulo podría afectar a todo el sistema.
   - Tiene una arquitectura basada en módulos.
   - El espacio de usuario es más complejo que el espacio del kernel.
   - Todas las respuestas son correctas.
3. **¿Qué beneficio ofrecen los sistemas operativos multiprocesador?**
   - La posibilidad de que se ejecuten varias tareas simultáneamente.

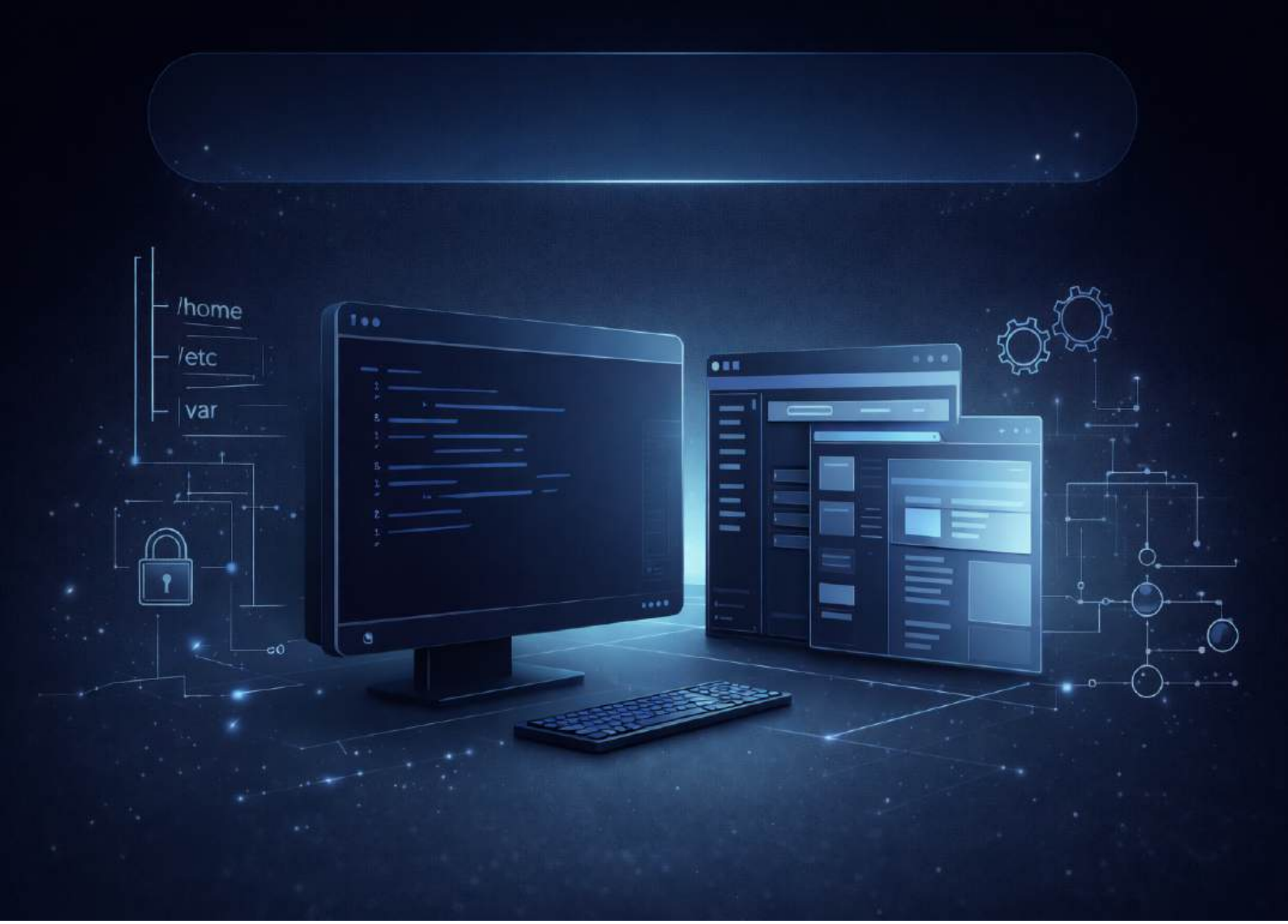
- La posibilidad de contar con varios núcleos y varios hilos por procesador.
- La posibilidad de que exista en el sistema programación paralela.
- La posibilidad de que exista en el sistema programación concurrente.

4. **¿En qué se diferencian principalmente los sistemas operativos distribuidos de los de red?**

   - Los sistemas operativos distribuidos hacen que varias computadoras parezcan un único sistema coherente, mientras que los de red facilitan la conexión y uso compartido de recursos sin integración de procesamiento.
   - Los sistemas distribuidos no permiten la interacción entre distintos nodos de la red.
   - Los sistemas operativos de red necesitan manejar perfiles de usuario.
   - Los sistemas operativos distribuidos se caracterizan por la transparencia.

5. **¿Cómo seleccionar un sistema operativo?**

   - Depende del tipo de dispositivo electrónico, para PC, Servidor, smart phone, dispositivo de red, etc.
   - Depende de los servicios y aplicaciones de la empresa.
   - Depende de los niveles de seguridad exigidos.
   - Ninguna respuesta es correcta.



CAPÍTULO IV

Explorando Linux

Introducción

Con el avance de los sistemas de código abierto, el uso de ***Linux*** se ha convertido en una alternativa fundamental dentro del mundo de la informática. Este sistema operativo se caracteriza por ofrecer flexibilidad, seguridad y libertad, permitiendo a los usuarios realizar desde tareas básicas hasta operaciones avanzadas en entornos de desarrollo, administración de servidores y uso cotidiano en computadoras personales.

Las distribuciones Linux son sistemas operativos instalables derivados del núcleo Unix, combinados con el Proyecto GNU y otros programas. Fueron agrupados en función de las necesidades específicas de cada usuario o empresa, sea de escritorio o servidor. Se caracterizan por su código abierto, modularidad y compatibilidad con gestores de paquetes, ofreciendo gran flexibilidad y opciones a los usuarios finales.

Dentro de las múltiples distribuciones, ***Ubuntu*** ocupa un lugar destacado por su facilidad de uso y accesibilidad. Su patrocinador es Canonical. Esta distribución tomó como base a Debian y se consolidó como una opción moderna, fácil de instalar y con actualizaciones regulares cada seis meses. Gracias a estas características, Ubuntu se ha posicionado como un sistema operativo confiable y ampliamente utilizado en entornos domésticos, académicos y empresariales.

Como aportación este capítulo ayuda al lector a saber cómo instalar y configurar un entorno de Linux (Ubuntu), tanto de forma gráfica (GUI) y por medio de línea de comandos (CLI) con la familiarización de los comandos básicos para un manejo adecuado del sistema.

Además, se presentarán actividades de aprendizaje, prácticas de laboratorio, problemas propuestos y resueltos, así como recursos de interés que facilitarán el fortalecimiento de su conocimiento.

I trust Linux for ever

¡Bienvenidos!

Fundamentos de Linux

Linux es un sistema operativo de código abierto basado en Unix que se distribuye bajo licencias libres, lo que permite a cualquier usuario estudiar, modificar y redistribuir su código fuente. Su desarrollo se realiza de manera colaborativa a través de comunidades y organizaciones en todo el mundo, lo que ha dado lugar a múltiples distribuciones adaptadas a distintos entornos [67].

Linux fue creado basado en los principios de la filosofía de libertad del usuario, propuestos por la Fundación de Software Libre (Free Software Foundation, FSF) y del licenciamiento público general (General Public License, GPL) [23, 104].

El sistema se organiza mediante una **estructura modular**, que permite gestionar usuarios, grupos y privilegios, así como incorporar módulos adicionales al kernel para ampliar sus funcionalidades [12]. A ello se suma la capacidad de **multitarea**, que posibilita la ejecución simultánea de múltiples procesos [124].

Linux también es un sistema **multiusuario**, lo que significa que se puede administrar varias cuentas en un mismo equipo, manteniendo independencia en los archivos y configuraciones de cada usuario [94].

Otro aspecto es su **sistema de archivos jerárquico**, organizado en forma de árbol con la raíz **"/"** como punto inicial. A partir de ella se despliegan directorios como **"/home"**, destinado a los archivos de los usuarios, y **"/etc"**, donde se almacenan las configuraciones del sistema. Esta estructura sigue el estándar Filesystem Hierarchy Standard (FHS.

En materia de **seguridad**, Linux implementa un modelo basado en permisos de lectura, escritura y ejecución para cada archivo o directorio, asignados a propietarios, grupos y otros usuarios [13].

Finalmente, Linux dispone de una **interfaz de línea de comandos (CLI)** que complementa las interfaces gráficas. A través de la CLI es posible instalar programas, administrar configuraciones y controlar servicios del sistema [110].

Distribuciones

Una de las principales características de Linux es la existencia de múltiples **distribuciones**, también conocidas como “distros”. Cada distribución combina el kernel de Linux con un conjunto de herramientas, entornos de escritorio, gestores de paquetes y configuraciones específicas. Esta diversidad es posible

gracias a la naturaleza de código abierto de Linux, que permite a comunidades y organizaciones crear versiones adaptadas a diferentes necesidades [67], [34], [137]. La Figura IV.1 muestra las las distribuciones más destacadas en el mundo, algunas de ellas que se explican brevemente a continuación:

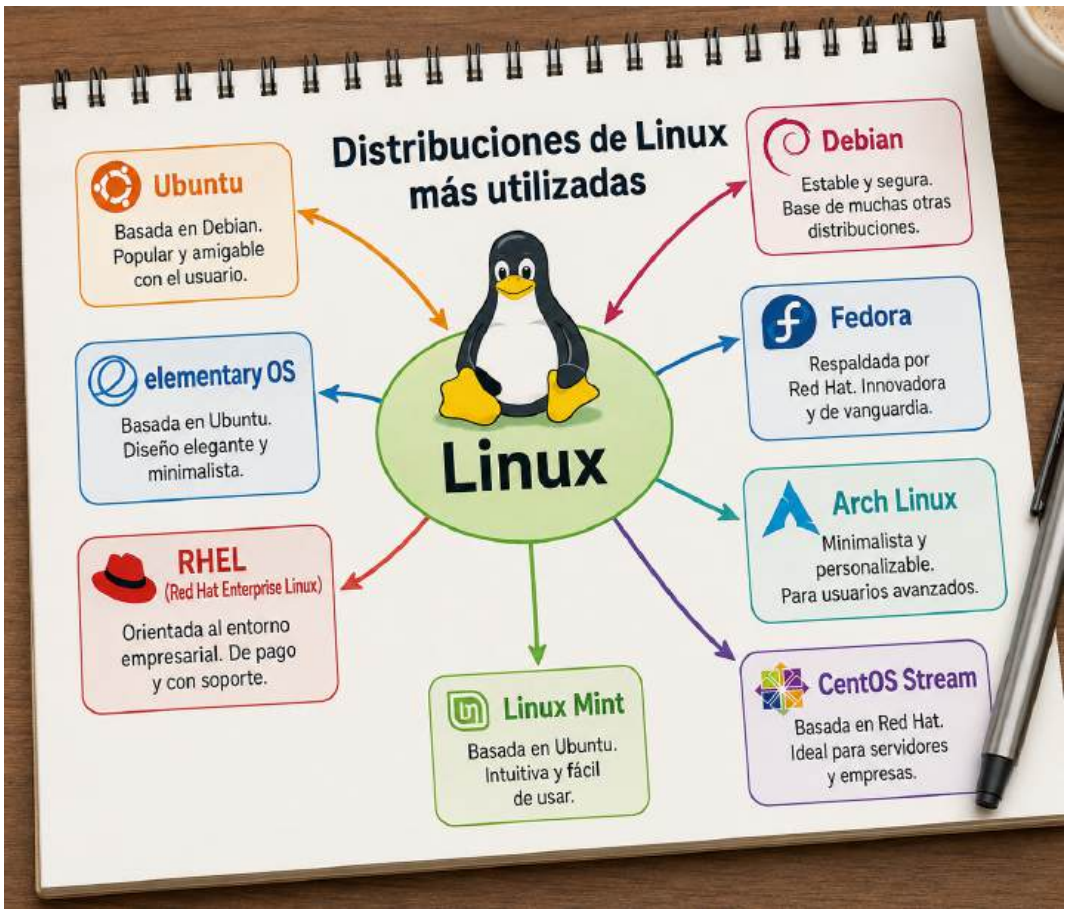


Figura IV.1: Mapa mental de las Distribuciones más utilizadas en el mundo. **Fuente:** Esta imagen fue creada por los autores utilizando Microsoft Copilot IA

Debian: cuyo lanzamiento fue en 1993, fue muy utilizada en España y Alemania. Utilizaba como entorno gráfico especialmente Gnome, muy segura, estable. Luego, basadas en Debian, se derivaron importantes distribuciones que se utilizan hoy en día, por el adecuado soporte técnico que reciben, como son Ubuntu para desktop y para servidor, así como Kali y Linux Mint.

Ubuntu: lanzada en 2004 por la empresa Canonical, está basada en Debian. Se diseñó para facilitar la instalación y el uso en equipos de escritorio, aunque actualmente es utilizada en servidores y entornos de nube. Canonical ofrece versiones con soporte a largo plazo (LTS) [22].

Fedora: patrocinada por Red Hat, es una distribución comunitaria que fun-

ciona como laboratorio de innovación. Sus desarrollos y características experimentales suelen servir de base para Red Hat Enterprise Linux (RHEL). [35].

Kali Linux: desarrollada por Offensive Security, está basada en Debian y orientada a la seguridad informática. Incluye de forma preinstalada un conjunto de herramientas para pruebas de penetración, auditorías de seguridad, análisis forense y hacking ético. Es utilizada en el ámbito académico, por investigadores de ciberseguridad y profesionales de pruebas de intrusión [92].

Proceso de instalación

La instalación de Ubuntu Linux en un entorno virtualizado constituye una alternativa para el aprendizaje y la experimentación con este sistema operativo. El uso de una máquina virtual permite evaluar y manipular el sistema en un entorno controlado, seguro y reversible.

Requisitos previos

Antes de iniciar el proceso de instalación, es necesario disponer de los siguientes elementos:

- **Oracle VirtualBox**, software de virtualización que puede descargarse de manera gratuita desde el sitio oficial: `https://www.virtualbox.org/`.
- La **imagen ISO de Ubuntu**, disponible en el portal oficial de descargas: `https://ubuntu.com/download/desktop`.
- Un computador con **4 o más GB en RAM** y **25 GB de espacio en disco recomendado por Virtual Box** libres.

Instalación de Ubuntu Desktop en Oracle VirtualBox

En este apartado se señala el procedimiento para crear una máquina virtual en VirtualBox e instalar la imagen .ISO de Ubuntu Desktop, edición Long Term Support (LTS) más reciente (*i.e, versión 22.04*).

1. **Creación de una nueva máquina virtual**
    - Iniciar Oracle VM VirtualBox y seleccionar la opción **Nueva** en la barra de herramientas.

- Asignar un nombre para la máquina virtual, por ejemplo, *UbuntuLTS*. Luego, establecer **Linux** como tipo de sistema operativo y seleccionar **Ubuntu (64-bit)** en la versión correspondiente.

2. **Asignación de recursos del sistema**

   - Configurar la cantidad de **memoria RAM** y el número de **CPU virtuales** que se asignarán a la máquina virtual, considerando la capacidad del equipo anfitrión.

3. **Creación del disco duro virtual**

   - Seleccionar la opción **Crear un disco duro virtual ahora** y establecer el tamaño de almacenamiento adecuado para la instalación. Se recomienda asignar un mínimo de 25 GB para asegurar el funcionamiento del sistema y la instalación de software adicional.

4. **Revisión de la configuración inicial**

   - Verificar que los recursos asignados (memoria RAM, CPU, almacenamiento y tipo de sistema operativo) sean correctos y cumplan con los requisitos mínimos establecidos para la instalación de Ubuntu, antes de finalizar la creación de la máquina virtual.

5. **Carga de la imagen ISO de Ubuntu**

   - Acceder al apartado **Almacenamiento** de la máquina virtual y seleccionar la opción para añadir un nuevo disco. A continuación, cargar la imagen **ISO de Ubuntu Desktop LTS** previamente descargada desde el sitio oficial.

6. **Inicio del proceso de instalación de Ubuntu**

   - Iniciar la máquina virtual y seleccionar la opción *Try or Install Ubuntu* en el menú de arranque. Posteriormente, seguir las instrucciones del asistente de instalación para completar el proceso.
   - Durante la instalación, el sistema realiza la copia de los archivos y la configuración inicial de los parámetros básicos del entorno operativo.

Interfaz de Línea de Comandos

La Interfaz de Línea de Comandos (CLI) es un componente esencial en Ubuntu y en cualquier distribución de Linux. A diferencia de las interfaces gráficas de usuario (GUI), la CLI permite al usuario interactuar directamente con el sistema operativo mediante la ejecución de comandos, lo que otorga un control más preciso sobre la administración, configuración y operación del sistema [100, 37]. La Figura IV.2 muestra la interfaz de línea de comandos de Ubunto.

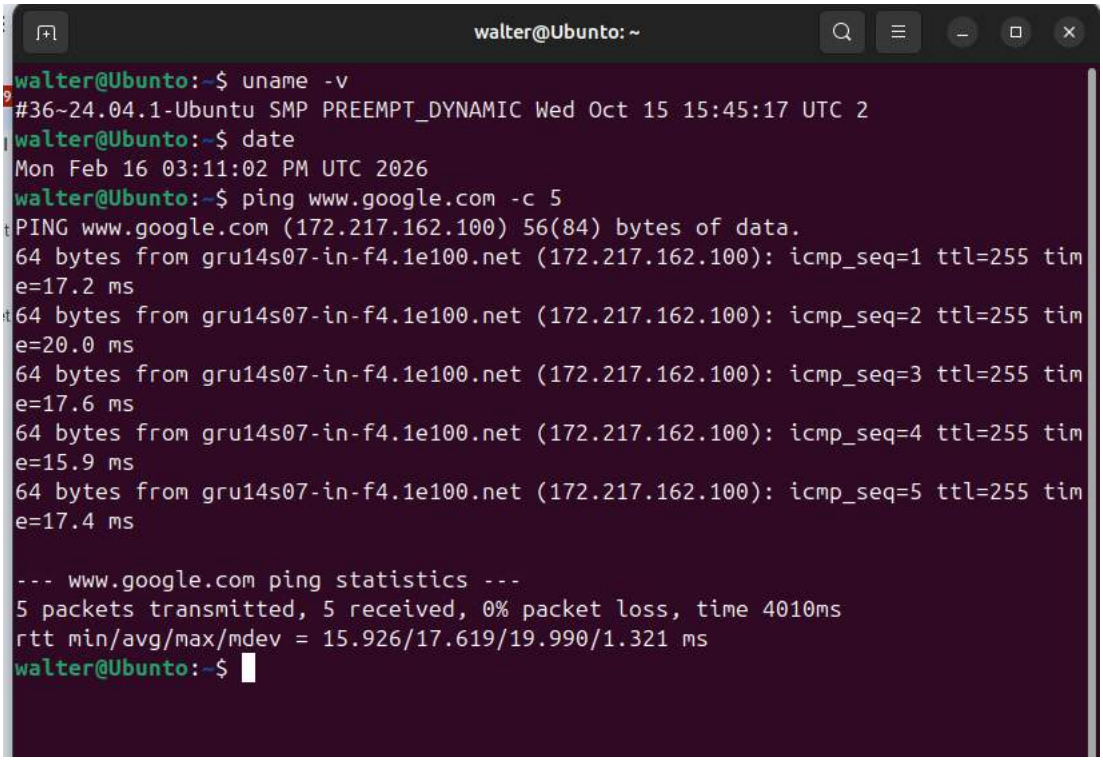


Figura IV.2: Interfaz gráfica de usuario en Ubuntu

Asimismo, la documentación oficial de Ubuntu menciona que la CLI resulta indispensable para tareas de mantenimiento, automatización y administración remota, debido a que requiere un menor consumo de recursos y facilita la ejecución de procesos avanzados [100, 37].

Acceso a la Terminal

Para abrir la terminal, se lo puede realizar de las siguientes formas:

- Con el teclado **Ctrl + Alt + T**.
- Dentro del menú, en la aplicación **Terminal**.

Al abrir la terminal, se muestra un entorno basado en texto en el cual el usuario introduce comandos para ejecutar tareas de administración del sistema, instalación de software, gestión de archivos, entre otras funciones.

Interfaz gráfica de usuario

La interfaz gráfica de usuario (GUI) es la representación visual del sistema operativo. Mediante íconos, ventanas y menús, la GUI ofrece un entorno intuitivo y accesible para el usuario. Este entorno permite la interacción con aplicaciones, archivos y configuraciones sin necesidad de emplear comandos en la terminal [124], como se muestra en la Figura IV.3.

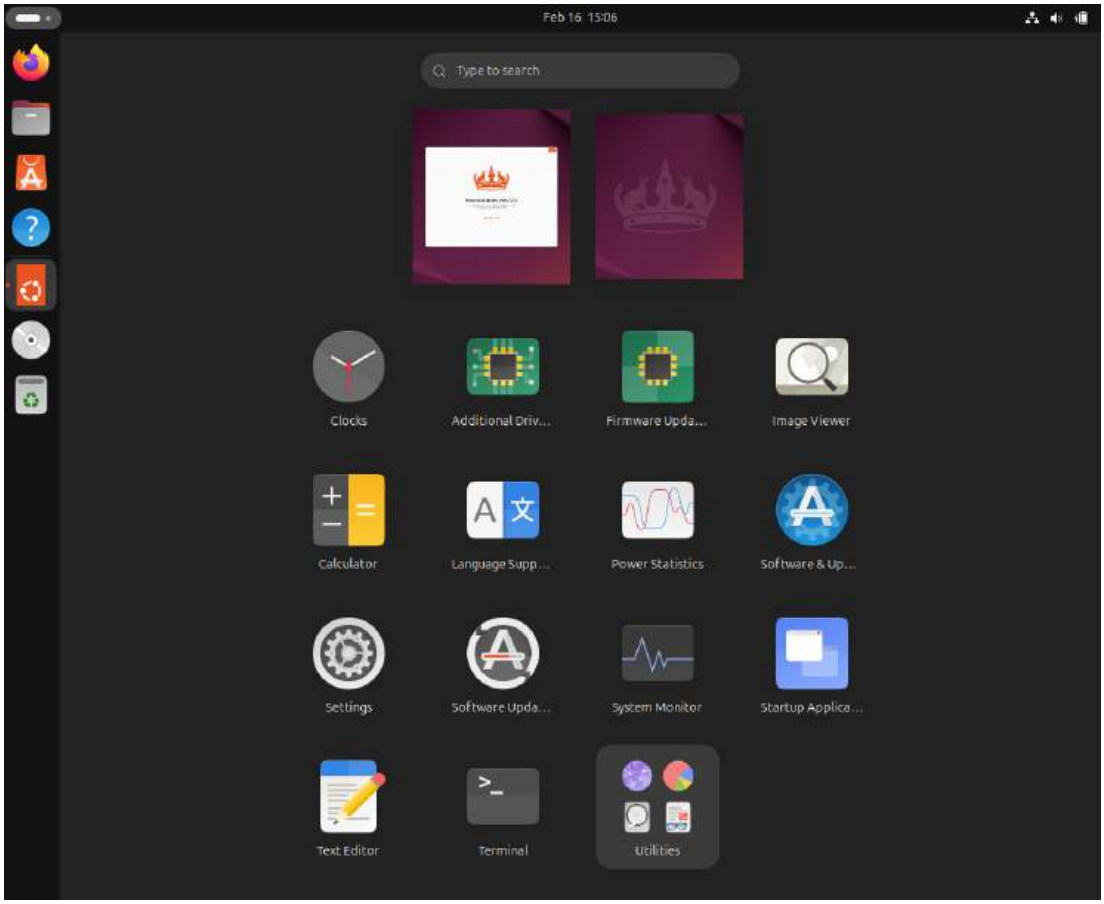


Figura IV.3: Interfaz gráfica de usuario en Ubuntu

Componentes principales de la GUI

A continuación, se describen algunos de los componentes de la interfaz gráfica de usuario dentro de Ubuntu:

- **Escritorio:** Área principal de trabajo para ubicar accesos directos a aplicaciones, archivos y carpetas. En Ubuntu, el entorno gráfico GNOME controla este espacio [131].
- **Barra de tareas (Dock):** Se ubica en el lateral izquierdo o en la parte inferior de la pantalla, esta facilita el acceso a aplicaciones frecuentes y al cambio entre ventanas abiertas.
- **Menú de aplicaciones:** Componente de la interfaz gráfica que organiza e inicia los programas instalados, herramientas de configuración y lugares del sistema.
- **Ventanas:** Cada aplicación utiliza una ventana con opciones para maximizar, minimizar, cerrar o cambiar su tamaño[124].
- **Área de notificaciones:** Localizada en la parte superior derecha. Muestra información del sistema, por ejemplo: hora, carga de batería, estado de la red y alertas de aplicaciones [131].
- **Menú de configuración:** Interfaz, donde se gestiona la configuración del sistema, aplicaciones, y parámetros del kernel.

Comandos básicos de Linux

La terminal de Ubuntu ofrece una amplia variedad de comandos para la administración del sistema. A continuación, se presentan algunos de los más utilizados:

- **pwd (Print Working Directory):** Indica la ruta absoluta del directorio donde se ubica el usuario dentro del sistema de archivos.

  Muestra el directorio actual

  ```
  pwd
  ```

- **sudo (Super User Do):** Permite a usuarios autorizados ejecutar programas con privilegios de seguridad de otro usuario, generalmente el administrador o root.

  Privilegios de superusuario

  ```
  sudo apt update
  ```

- **apt (Advanced Package Tool):** Herramienta de gestión de paquetes predeterminada, la cual permite instalar, actualizar y eliminar programas desde la terminal.

Gestión de paquetes con `apt`

```
sudo apt install paquete # Instala un paquete
sudo apt update # Actualiza la lista de paquetes
sudo apt upgrade # Actualiza el software
instalado
```

- **man (Manual):** Interfaz del sistema para acceder a las páginas del manual de referencia, es decir, la documentación oficial.

Manual del comando

```
man ls # Muestra el manual del comando 'ls'
```

- **uname:** Indica información del sistema operativo, como el nombre del kernel, la versión instalada y la arquitectura del procesador.

Comando para ver la información del sistema

```
uname -a
```

- **whatis:** Presenta una descripción breve sobre la función de un comando disponible en el sistema.

Descripción de comandos

```
whatis ls
```

- **info:** Proporciona información detallada sobre un comando o programa.

Información detallada de un comando

```
info ls
```

- **–help:** Muestra una ayuda sobre el uso de un comando.

Comando de ayuda

```
ls -help
```

- **clear:** Limpia la pantalla de la consola, elimina el historial previo.

Comando para limpiar la terminal

```
clear
```

- **date:** Indica la fecha y hora actuales configuradas en el sistema.

Fecha y hora actuales

```
date
```

- **whoami:** Presenta el nombre del usuario actual que inició sesión en el sistema.

Nombre de usuario actual

```
whoami
```

- **>:** Redirige la salida estándar a un archivo, creando uno nuevo o sobrescribiendo el existente.

Redirigir la salida estándar

```
ls >archivo.txt
```

- **cat:** Sirve para visualizar, combinar (concatenar) y crear archivos de texto rápidamente desde la terminal

Comando para ver el contenido de un archivo

```
cat archivo.txt
```

- **less:** Visualizador de archivos de texto diseñado para leer archivos grandes, permite navegar por páginas, realizar búsquedas y abrir documentos sin carga en memoria.

Comando para navegar en un archivo

```
less archivo.txt
```

- **tree:** Muestra la estructura de directorios en forma de árbol.

Comando para ver estructura de directorios

```
tree
```

- **history:** Muestra el historial de comandos ejecutados en la sesión actual.

Historial de comandos

```
history
```

Apache OpenOffice

Apache OpenOffice es una suite ofimática de código abierto que incluye aplicaciones para redactar textos, gestionar hojas de cálculo, crear presentaciones, diseñar gráficos y administrar bases de datos. Es compatible con diversos sistemas operativos como Windows, macOS y Linux, y se distingue por ser libre y por soportar formatos de archivo comunes [29].

El sitio web oficial de LibreOffice es: `https://es.libreoffice.org/`. Es un software sin fines de lucro que mantiene la filosofía de las libertades del usuario establecidas en la Fundación de Software Libre. LibreOffice mantiene una licencia dual LGPLv3 dado que es software libre. En la Figura IV.4 se visualiza el logo de LibreOffice.

Figura IV.4: Logo de Apache OpenOffice.

Componentes de Apache OpenOffice

Los principales componentes de la suite Apache OpenOffice se presentan en la Tabla IV.1. Además, en la Figura IV.5 se muestra el **LibreOffice Calc** que consiste en una aplicación de similares funcionalidades que Excel de Microsoft Office [7].

Tabla IV.1: Componentes de Apache OpenOffice y sus equivalentes

Aplicación	Descripción	Equivalente en Microsoft Office
Writer	Procesador de textos. Permite editar documentos, usar plantillas, tablas y exportar a PDF [29].	Word
Calc	Hoja de cálculo para realizar operaciones numéricas, gráficos y manejo de datos [29].	Excel
Impress	Aplicación para presentaciones, con diapositivas, efectos y transiciones.	PowerPoint
Draw	Dibujo e ilustración [29].	Visio
Base	Sistema de gestión de bases de datos relacional [29].	Access
Math	Editor de ecuaciones matemáticas que se usa dentro de otros documentos [29].	Editor de ecuaciones de Word

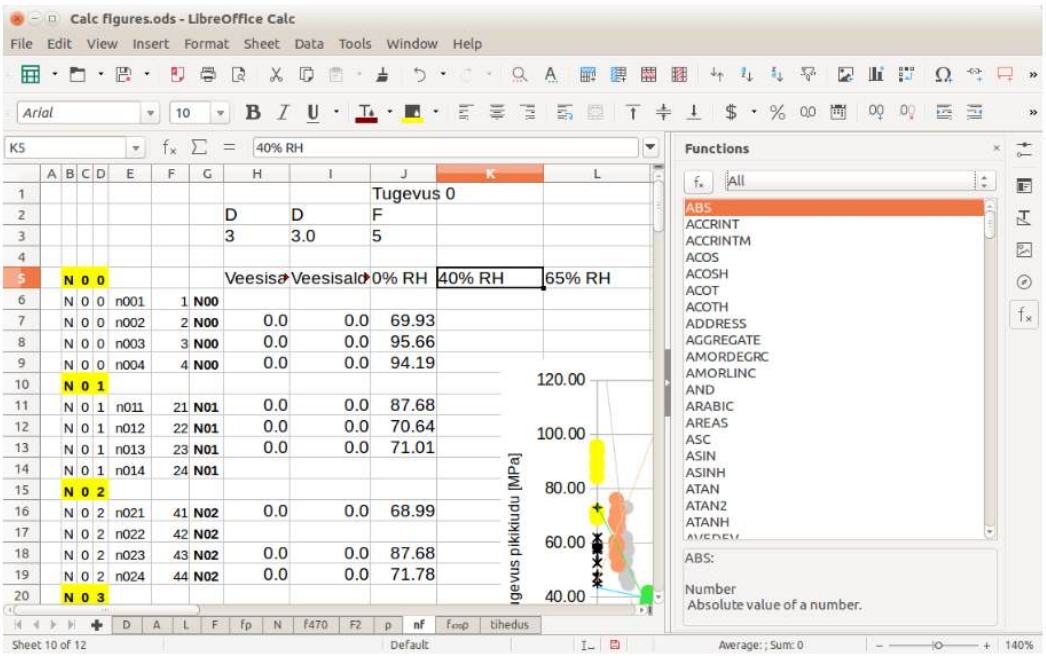


Figura IV.5: Ejemplo de un informe financiero editado y procesado con LibreOffice Calc.

Conclusiones y Lecciones Aprendidas

En este capítulo se ha examinado el uso de **Linux**, con un énfasis en la distribución **Ubuntu**. Se destacaron las características y ventajas que la convierten en una opción adecuada para usuarios que inician su aprendizaje en sistemas operativos de código abierto. Ubuntu se presenta como una alternativa, dado

a que requiere un consumo moderado de recursos de la máquina anfitriona, ofrece un proceso de instalación sencillo y proporciona herramientas esenciales para la administración tanto gráfica como mediante comandos en la línea de terminal.

Las distribuciones de Linux reúnen una amplia gama de librerías y paquetes para levantar servicios y aplicaciones, para gestionar recursos, conectividad, seguridad de la información, usuarios, controles de acceso, entre otros tanto en forma individual como para las empresas. Su aprendizaje es muy útil para la vida profesional de los técnicos, ingenieros y especialistas de las empresas.

Lecciones Aprendidas

Ubuntu Linux es una plataforma flexible y segura, adecuada tanto para usuarios principiantes como para usuarios con mayor experiencia. Esto se debe a su organización interna, su sistema de permisos y su capacidad para ejecutar varias tareas al mismo tiempo.

La **interfaz de línea de comandos (CLI)** es una herramienta útil porque permite un control directo del sistema. Mediante comandos como `ls`, `cd`, `sudo` y `apt`, se pueden administrar archivos, gestionar permisos e instalar programas de forma rápida y eficiente.

El **sistema de archivos** en Linux tiene una estructura ordenada en forma de árbol, que comienza en la raíz `/` y se extiende a directorios como `/home` y `/etc`, los cuales ayudan en la gestión de usuarios y la configuración del sistema.

El uso de **máquinas virtuales** para instalar y probar Ubuntu es una estrategia para el aprendizaje, ya que permite practicar y conocer el entorno Linux sin afectar el sistema operativo principal del equipo.

Sitios de interés

(I) Video con los pasos para la descarga de Ubuntu explicando los detalles de la GPU, RAM y Espacio necesario: Instalar Ubuntu en VirtualBox - Windows 10/11

(II) Lista de Comandos Básicos para practicar en la terminal de Ubuntu: Los 60 comandos básicos de Linux, página Web que muestra diversos comandos para quienes estan iniciando en el mundo de Linux.

Autoevaluación del Capítulo

1. **¿En qué se diferencia Ubuntu con un sistema operativo de paga?**
   - Solo los desarrolladores pueden modificar el código.
   - Cualquiera puede modificar y distribuir el código.
   - Es un software comercial.
   - Solo las empresas pueden utilizarlo.
2. **¿Qué distribución de Linux sirve como base para Ubuntu y Linux Mint?**
   - Fedora
   - Arch Linux
   - Debian
   - Gentoo
3. **¿Qué comando muestra el historial de comandos ejecutados?**
   - ls
   - sudo su
   - history
   - mkdir
4. **¿Qué ventaja tiene la interfaz de línea de comandos (CLI) en Ubuntu frente a la interfaz gráfica?**
   - Solo permite cambiar el aspecto del escritorio.
   - Permite automatizar tareas mediante scripts.
   - No se puede usar para instalar programas.
   - Solo funciona en modo seguro.
5. **¿Qué componente de la interfaz gráfica de usuario (GUI) de Ubuntu está diseñado específicamente para buscar y abrir aplicaciones instaladas en el sistema?**
   - Escritorio
   - Barra de Tareas (Dock)

- Menú de Aplicaciones
- Área de Notificaciones

6. **¿Qué distribución de Linux está diseñada específicamente para pruebas de penetración y seguridad informática?**
   - Ubuntu
   - Kali Linux
   - Fedora
   - CentOS
7. **¿Cual de los siguientes comandos tiene la funcionalidad de cambiar entre directorios?**
   - ls
   - cd
   - mv
   - docker ps
8. **¿Cuál es la función del comando 'sudo' en Linux?**
   - Ejecutar un comando con privilegios de administrador.
   - Crear un nuevo archivo.
   - Listar archivos en un directorio.
   - Cambiar de directorio.
9. **¿Qué herramienta de OpenOffice tiene la funcionalidad de realizar hojas de cálculos?**
   - Writer
   - Calc
   - Draw
   - Math
10. **¿Qué directorio contiene los archivos de configuración del sistema en Linux?**
    - /home

- /etc
- /usr
- /var

CAPÍTULO V

Programación en Shell Script

Introducción

Tal como se había señalado en capítulos anteriores; la Shell como parte del Sistema Operativo cumple dos funciones: **La primera**, es una interfaz de línea de comandos (Command Line Interface (CLI)) basada en texto, la cual permite ingresar órdenes e instrucciones que interactúan con el sistema operativo de una computadora. **La segunda**, es un lenguaje de programación intérprete que contiene órdenes escritas (shell scripts) con una sintaxis y semántica determinada, que permite automatizar y ejecutar programas, tareas u órdenes desde la línea de comandos. Este apartado se concentra en el **Lenguaje de programación shell script**.

Los **shells script** son utilizados por los administradores de los sistemas computacionales ya que permiten, con una sola orden, ejecutar múltiples comandos. Los scripts se utilizan para automatizar tareas tales como realización de copias de seguridad, actualización del sistema, comprobación del espacio libre de disco, descarga de archivos, limpieza de archivos temporales, entre otros.

Como tal, un shell script se traduciría como un guión de órdenes, el cual es un archivo de texto que contiene órdenes interpretables por la interfaz de línea de comandos.

Este capítulo aporta en su formación, principalmente porque este lenguaje de programación, le permite automatizar tareas repetitivas para incrementar la eficiencia y porque reduce el riesgo de errores humanos [122]. Además, porque le otorgará una comprensión de cómo interactuar con el ***sistema operativo*** y manejar el CLI, comandos, procesos y archivos [48].

Al finalizar este capítulo, el lector comprenderá los fundamentos básicos de la programación en shell script. Aprenderá la estructura básica de un programa en shell, su edición y ejecución. Entenderá cómo funcionan los operadores aritméticos y de comparación, cómo declarar variables, las variables de entorno, las estructuras de programación, y cómo utilizar funciones en shell.

En este capítulo el lector también encontrará actividades de aprendizaje, problemas propuestos, resueltos, y sitios de interés que facilitarán la apropiación de su conocimiento.

Como plataforma de entrenamiento que permitirá realizar las pruebas de concepto y formalizar el aprendizaje, se utilizará la interfaz de línea de comandos de una distribución de Linux tanto para desktop como para servidor.

¡Bienvenidos!

Fundamentos de programación shell

Para iniciar este capítulo, es preciso destacar el estándar denominado **Interfaz de Sistema Operativo Portátil basado en UNIX**, *POSIX*, que es una norma escrita por la IEEE que define una interfaz estándar del sistema operativo y su entorno vinculado al intérprete de comandos Shell Script [48]. POSIX fue sugerido por Richard Stallman en 1980, como respuesta a la demanda del IEEE, que buscaba un nombre fácil de recordar [122].

Tipos de shell script

Sobre la base del estándar de Portable Operating System Interface UNIX, en los sistemas operativos Unix 7 en 1979, se introdujeron varias implementaciones de Shell tales como:

- **sh (Bourne Shell):** desarrollado por Stephen Bourne de los Laboratorios Bell de AT&T y que fue empleado desde las primeras versiones de Unix (i.e., Unix Versión 7). En algunas distribuciones de Linux se se localiza en la carpeta /bin/sh, del árbol de directorios de Linux. Cabe señalar que sobre sh, se han el zsh (Z shell), ash (almquist shell), bash (Bourne again shell), dash (Debian almquist shell) y ksh (Korn shell), algunos de ellos que se listan a continuación.

- **bash (Bourne Again Shell):** que incluye la funcionalidad del sh, ksh y csh, permitiendo el desarrollo de los comandos asignados por defecto a los usuarios, considerando shell lo más empleado en la mayoría de las consolas de comandos de Linux [135]. En algunas distribuciones se localiza en la carpeta /bin/bash, del árbol de directorios de Linux

- **dash (Debian almquist shell)** , se caracteriza por ser mucho más ligero, al depender de menos bibliotecas, lo que le hace más eficiente que bash, aunque con menos características funcionales.

- **ksh (Korn shell):** se destaca por el manejo de archivos, pudiendo competir con lenguajes de programación especializados tales como awk o perl.

- **csh (C shell)** : maneja una sintaxis muy parecida al lenguaje de programación C, sin embargo, es menos utilizado respecto a sh y sus derivados analizados en este apartado [20].

Estructura básica de un programa en Shell-script

Para crear un programa en Shell-script, se debe utilizar un editor de texto plano como `vi`, `vim` o `nano`.

Creación de un programa en Shell Script

```
vim nombre del programa.sh
```

Todo programa de Shell-script es un archivo de texto que contiene varios comandos e instrucciones. Tiene un nombre y regularmente una extensión .sh.

La estructura básica de un programa en Shell-script sería la siguiente:

```
#!/bin/bash
# Inicio Programa.
# Esto es un comentario y no se interpreta
# Echo "declaraci n de variables globales"
# Echo "declaraci n de funciones"
function hola(){
    echo "Hello world"
    # desde la funci n hola
    # Echo "Declaraci n de variables locales"
}
mensaje(){
    echo "I love Linux"
    # Desde la funci n saludo.
    # Declaraci n de variables locales
}
# Llamada a las funciones
hola
mensaje
# Fin Programa
```

Listing V.1: Estructura básica de un programa en Shell-script

En donde: La primera es una línea especial al inicio de un script, que se conoce como **shebang**, que indica al sistema operativo qué intérprete debe utilizar para ejecutar el script.

La segunda línea es un comentario que no será interpretado.

A partir de la tercera línea del script pueden utilizarse varios comandos, declarar las variables globales, declarar funciones, que incluyen estructuras de control, comentarios y llamadas a funciones.

Para ejecutar un programa de Shell script se pueden utilizar los siguientes comandos desde la interfaz de línea de comandos, seguido del nombre el programa.sh:

Cómo ejecutar un programa en Shell Script

```
sh nombre del programa.sh
```

Cómo ejecutar un programa en Shell Script

```
source nombre del programa.sh
```

Cómo ejecutar un programa en Shell Script

```
bash nombre del programa.sh
```

Cómo ejecutar un programa en Shell Script

```
./nombre del programa.sh
```

Declaración de variables

En Shell script, las variables pueden contener cualquier tipo de datos tales como cadenas, números enteros, arreglos, etc. Como se indicó en apartados anteriores, en shell script, no existe un tipo de dato numérico específico. Tampoco, hay un tipo booleano explícito. Sin embargo, los tipos de datos más comunes son las cadenas de texto, cadena de caracteres (i.e., strings) o datos alfanuméricos.

Las variables en shell script se declaran como **NOMBRE=valor** y su valor se utiliza anteponiendo el símbolo "$"delante del nombre de la variable.

Variables en Shell Script

```
NOMBRE=Ricardo
APELLIDO=Rivadeneira
Contador=10
Acumulador=100
MENSAJE="Hello World"
Mensaje=”I love Linux”
$NOMBRE
$APELLIDO
$Contador
$Acumulador
$MENSAJE
$Mensaje
```

Operadores aritméticos

En Shell Script, los operadores aritméticos permiten realizar cálculos dentro de los programas para manipular datos numéricos. Puesto que Shell Script no soporta operaciones aritméticas directamente, se utiliza el comando **expr**, la calculadora básica **bc** (basic calulator) u otras expresiones aritméticas para realizar estas operaciones. Los operadores aritméticos básicos se los lista en la Tabla V.1:

Tabla V.1: Operadores Aritméticos

Operación	Operador	Descripción
Suma	+	Sumar dos números.
Resta	-	Restar dos números.
Multiplicación	*	El producto de dos números.
División	/	Divide el dividendo del divisor y devuelve solo la parte entera del cociente.
Módulo	%	El residuo de la división del primer operando por el segundo.
Potenciación	**	Obtiene la potencia sobre la base.
Incremento	++	Aumenta en una unidad el valor del operando.
Decremento	- -	Reduce en una unidad el valor del operando.

Operadores de comparación

Estos operadores booleanos evalúan dos valores y devuelven Verdadero equivalente a 1 o Falso, que es igual a 0. Los operadores de comparación se los lista en la Tabla V.2:

Tabla V.2: Operadores de comparación

Comparador	Operador	Descripción
<	-lt	Less than
<=	-le	Less and equal
>	-gt	Great than
>=	-ge	Great and equal
<>	-ne	Not equal
=, ==	-eq	Equal
Conjunción	-&&	And, -a
Disjunción	-||	Or, -o

Variables en Entorno Linux

Son valores dinámicos que almacenan información del entorno del sistema operativo. Son marcadores de posición para la información almacenada que

pasa datos a los programas iniciados en su intérprete de comandos. Suelen ser escritas en letras mayúsculas, aunque también pueden crearse variables de entorno en minúsculas. Estas variables se pueden crear, editar, guardar y eliminar. Una lista completa de variables de entorno de su distribución de Linux puede visualizarse utilizando el siguiente comando:

Cómo listar las variables de entorno

```
printenv | less
```

Cada línea contiene el nombre de la variable de entorno Linux seguido de = y el valor. Por ejemplo: **HOME=/home/aymé**. HOME es la variable de entorno de Linux que tiene el valor establecido como el directorio **/home/aymé** en su cuenta de usuario.

A continuación, en el siguiente programa en shell script se imprimen en pantalla las variables de entorno en Linux:

Ejercicio 1: Variables de entorno

Descripción: Este programa imprime las variables de entorno del sistema.

```
#!/bin/bash
#Programa: Variables de entorno
clear
echo "Variables de entorno"
echo "El directorio actual es $PWD"
echo "La cuenta de usuario es $HOME"
echo "El tipo de shell es $SHELL"
echo "El usuario es $USER"
echo "La terminal es $TERM"
echo "El idioma, codificaci n y localizaci n $LANG"
echo "I love Linux, bye"
#Fin Programa: Variables de entorno
```

Listing V.2: Variables de Entorno Linux

Estructuras de programación

Estructuras Condicionales

Las estructuras condicionales permiten ejecutar una acción si se cumple con una condición. Habilita decidir mediante operadores de comparación, cuál será el flujo del programa sobre la base del resultado de la evaluación de una condición.

Sentencia If

En Shell script, se utilizan normalmente la estructura **if else**, que evalúa la condición. Si es verdadera, ejecutará una sentencia o un conjunto de sentencias que cumplen la condición. En caso de que sea falsa, evaluará la siguiente condición. El formato de la sentencia **if** es la siguiente:

```
#!/bin/bash
if [ condition 1 ]; then
    echo "Sentencias por verdadero."
elif [ condition 2 ]; then
    echo "Sentencias por verdadero."
else
    echo "Sentencias por falso o caso contrario."
fi
```

Estructuras de repetición

Son utilizadas para repetir un conjunto de instrucciones u órdenes hasta que se cumpla la condición establecida. Se utiliza principalmente cuando no se conoce el número de veces que el bucle debe repetirse. La condición es comprobada antes de cada iteración. El formato de la sentencia **while** es la siguiente:

Sentencia While

```
#!/bin/bash
while [ condition ]; do
    echo "Conjunto de sentencias que cumplen la condicion"
done
```

Sentencia For

El bucle **for** permite la iteración o repetición de una lista de elementos uno por uno. En cada ciclo del bucle, se asigna a la variable un valor de la lista, cadena o rango, en el orden determinado, desde el primero hasta el último. A diferencia del bucle while, la condición es comprobada antes de cada iteración.

```
#!/bin/bash
for ((i = 0 ; i < 100 ; i++)); do
  echo $i;
  echo   Conjunto   de sentencias que se repiten
done
```

Funciones en shell script

En programación, es aconsejable fragmentar un programa en varias funciones, métodos o procedimientos. En este apartado se explican cómo funcionan las funciones y algunos conceptos que permitirán entenderlas.

En Shell script, una función es un bloque de código que contiene una secuencia de órdenes, instrucciones o comandos. Estas órdenes cumplen con una tarea específica de una aplicación que la invocó. Las funciones ayudan a evitar la repetición de código y a descomponer programas complejos en tareas más simples. Una función puede definir opcionalmente parámetros de entrada que permiten recibir argumentos, y también puede devolver un valor como salida. Una función puede ser llamadas varias veces. Las funciones en shell script, operacionalmente, son limitadas.

Declaración de funciones

En shell-cript al menos existen dos formas de declarad funciones

Primera forma de declarar una función

```
Nombre_función()
{
echo "Hola Mundo";
}
```

Otra opción para declarar una función es utilizando la palabra clave function. De esta manera, la función tendría el siguiente aspecto:

Segunda forma de declarar una función

```
function Nombre_función()
{
echo "Hola Mundo"
}
```

Ejemplos

```
1 #!/bin/bash
2
3 Hola_mundo() {
4    echo "Hello World"
5 }
6
```

```
Hola_mundo
```

Listing V.3: Funciones en shell script

```
#!/bin/bash

suma() {
   suma=$(expr $num1 + $num2)
   echo "La suma es: $suma"
}

resta() {
   resta=$(expr $num1 - $num2)
   echo "La resta es: $resta"
}

producto() {
   producto=$(expr $num1 \* $num2)
   echo "El producto es: $producto"
}

division() {
   division=$(expr $num1 / $num2)
   echo "La division es: $division"
}

residuo() {
   residuo=$(expr $num1 % $num2)
   echo "El residuo es: $residuo"
}

echo "Ingrese dos numeros enteros positivos:"
read num1
read num2

suma
resta
producto
division
residuo
```

Listing V.4: Funciones invocando operaciones aritméticas

```
#!/bin/bash
factorial() {
    local num=$1
    local resultado=1
    for (( i=2; i<=num; i++ )); do
        resultado=$((resultado * i))
    done
    echo $resultado
}
echo "Ingresa un numero entero positivo: "
read numero
if [[ $numero -lt 0 ]]; then
    echo "Se deben ingresar numeros enteros positivos."
else
```

```
    fact=$(factorial $numero)
    echo "El factorial de $numero es: $fact"
fi
```

Listing V.5: Función para calcular el factorial

Ejercicios resueltos

Ejercicio 1: Conversión de temperatura

Descripción: Este programa convierte temperaturas entre Celsius y Fahrenheit. El usuario debe ingresar la temperatura y la unidad de origen por teclado.

```
#!/bin/bash
# Programa: Conversion de temperatura

clear
echo "Conversion de temperatura"

echo "Ingrese la temperatura a convertir:"
read temp

echo "Seleccione la unidad (C para Celsius, F para Fahrenheit):"
read unit

if [ "$unit" == "C" ]; then
    fahrenheit=$(echo "scale=2; ($temp * 9/5) + 32" | bc)
    echo "$temp  C  es igual a $fahrenheit  F "
elif [ "$unit" == "F" ]; then
    celsius=$(echo "scale=2; ($temp - 32) * 5/9" | bc)
    echo "$temp  F  es igual a $celsius  C "
else
    echo "Unidad no v l i d a ."
fi
```

Listing V.6: Conversión de temperatura

Ejercicio 2: Tablas de multiplicar de un número ingresado por teclado

Descripción: En este programa el usuario ingresa un número por teclado y se imprime su tabla de multiplicar hasta el 20.

```
#!/bin/bash
#Programa: Tabla de multiplicar
clear
echo "Tablas de multiplicar de un n m e r o ingresado por teclado"
echo "Ingrese un numero entero positivo:"
read numero
a=1; b=0;
while [ $a -le 20 ]; do

  b=`expr $numero \* $a`
```

```
  echo "$numero * $a = $b"
  a=`expr $a + 1`
done
echo "I love Linux, bye"
#Fin Programa: Tabla de multiplicar
```

Listing V.7: Tablas de multiplicar de un número ingresado por teclado

Ejercicio 3: Multiplicación de dos números

Descripción: En este programa el usuario ingresa dos números y se imprime su producto, sin utilizar el operador de multiplicación.

```
#!/bin/bash
#Programa: Multiplica dos numeros mediante sumas susesivas
clear
echo "Multiplicacion de dos numeros ingresados por teclado"
echo "Ingrese el primer numero: "
read numero1
echo "Ingrese el segundo numero: "
read numero2
contador=1
resultado=$numero1
while [ $contador -lt $numero2 ]; do
  resultado=`expr $resultado + $numero1`
  contador=`expr $contador + 1`
done
echo "$numero1 * $numero2 = $resultado"
echo "I love Linux, bye"
#Fin Programa: Multiplicacion de dos numeros mediante sumas
    susesivas
```

Listing V.8: Multiplicación mediante sumas susesivas de dos números ingresados por teclado

Ejercicio 4: Sentencia 'for' con números

Descripción: En este programa se imprimen números del 1 al 10 usando el bucle 'for'.

```
#!/bin/bash
#Programa: Sentencia for con numeros
echo "Sentencia for con numeros"
for numero in {1..10}; do
  echo "$numero"
done
echo "I love Linux, bye"
#Fin Programa: Sentencia for con numeros
```

Listing V.9: Bucle for para lista de números

Ejercicio 5: Cateto de un triángulo

Descripción: Para este programa se pedirá que el usuario ingrese por teclado la longitud del cateto e hipotenusa de un triángulo y se calculará su segundo cateto.

```
#!/bin/bash
#Program: Calcular un cateto en base a la hipotenusa y el otro
   cateto
clear
echo "Teorema de pitagoras"
echo "Ingrese la longitud del cateto: "; read cateto
echo "Ingrese la longitud de la hipotenusa: "; read hipotenusa
if [ $cateto -ge $hipotenusa ]; then
  echo "La longitud de la hipotenusa debe ser mayor a la longitud
    del cateto."
else
  catetoR=$(echo "scale=2; sqrt(($hipotenusa^2) - ($cateto^2))" | bc
    )
  echo "La longitud del segundo cateto es: $catetoR"
fi
echo "I love Linux, bye"
#Fin Programa: Calcular un cateto en base a la hipotenusa y el otro
   cateto
```

Listing V.10: Cálculo del cateto de un triángulo

Ejercicio 6: Área de un círculo

Descripción: Para este programa el usuario ingresará el radio de un círculo y se calculará su área.

```
#!/bin/bash
#Program: Calcular el area de un c rculo
clear
echo " rea  de un circulo"
echo "Ingrese la longitud del radio: "; read radio
pi=$(echo "4*a(1)" | bc -l)
area=$(echo "scale=2; $pi * ($radio^2)" | bc)
echo "El area del circulo es: $area"
echo "I love Linux, bye"
#Fin Programa: Calcular el area de un c rculo
```

Listing V.11: Calcular el área de un circulo

Ejercicio 7: Serie de Fibonacci

Descripción: La serie de Fibonacci es una sucesión infinita de números naturales tales como: 0, 1, 1, 2, 3, 5, 7, 12, 19..... Para este programa el usuario ingresará el número límite de esta serie.

```
#!/bin/bash
#Program: Serie Fibonacci
clear
echo "Serie Fibonacci"
echo "Ingrese el numero limite para la serie: "; read numero
suma=0; n=0;n_1=1
for i in $(seq 1 $numero); do
  suma=$(echo "$n+$n_1" | bc)
  echo -n "$suma, "
  if [ `expr $i % 2 ` -eq 0 ]; then
    n=$suma
```

```
  else
    n_1=$suma
  fi
done
echo "..."
```

Listing V.12: Serie Fibonacci

Ejercicio 8: Factorial de un número

Descripción: Se representa por el signo de exclamación "!" y se calcula al multiplicar todos los números enteros y positivos que hay entre el número que aparece en la fórmula y el número 1. Por ejemplo: 5! = 1 * 2 * 3 * 4 * 5 = 120. Para este programa el usuario ingresará el número entero positivo del cual se calculará su número factorial.

```
#!/bin/bash
#Program: Factorial de un numero con BC
clear
echo -n "Ingrese un numero: "
read nu
fact=1
if [ $nu -eq 1 -o $nu -eq 0 ];then
        fact=1
else
        for i in $( seq 1 $nu ); do
                fact=$(echo "$fact*$i" | bc)
        done
fi
echo "$nu! = $fact"
```

Listing V.13: Serie Fibonacci

Ejercicio 9: Factorial de un número

Descripción: El factorial de un número se calcula al multiplicar todos los números enteros positivos desde 1 hasta n!. Por ejemplo: 5! = 1 * 2 * 3 * 4 * 5 = 120. Para este programa el usuario ingresará el número entero positivo del cual se calculará su número factorial.

```
#!/bin/bash
#Program: Imprimir sucesiones
clear
echo "Defina un rango:"
read ran
echo "Defina un limite:"
read lim
echo "Defina un inicio:"
read ini
for i in $(seq 0 $lim);do
        suc=$(echo "$ini+$ran*$i" | bc)
        echo -n "$suc  "
done
```

Listing V.14: Serie Fibonacci

Conclusiones y Lecciones aprendidas

Este capítulo ha sido enfocado en comprender los fundamentos básicos la programación en shell script. Los lectores han aprendido a estructurar un programa en bash, comenzando por la declaración de variables y el uso de variables de entorno, lo que permite gestionar el entorno de ejecución. Se ha explorado operadores de comparación y aritméticos, esenciales para realizar cálculos y tomar decisiones dentro del script. Se ha evidenciado que con el uso de cualquier estructura de programación se puede resolver cualquier problema computacionalmente, Además, mediante la declaración de funciones se ha permitido organizar el código y reutilizar la lógica, facilitando la resolución de problemas. La práctica a través de ejercicios propuestos y resueltos ha sido fundamental para consolidar estos conceptos, mejorando la capacidad para desarrollar scripts funcionales y efectivos.

Lecciones aprendidas

Los scripts se utilizan para automatizar tareas tales como realización de copias de seguridad, actualización del sistema, comprobación del espacio libre de disco, descarga de archivos, limpieza de archivos temporales, entre otros.

POSIX es una norma escrita por la IEEE que define una interfaz estándar del sistema operativo y su entorno vinculado al intérprete de comandos Shell Script.

Bourne Again Shell (bash) que incluye la funcionalidad del sh, ksh y csh, es el intérprete de comandos que viene predefinido en la consola de Linux.

En Shell Script, no existe un tipo de dato numérico específico. Tampoco, hay un tipo booleano explícito. Sin embargo, los tipos de datos más comunes son las cadenas de texto, cadena de caracteres (i.e., strings) o datos alfanuméricos.

Las variables en entorno Linux son valores dinámicos que almacenan información del entorno del sistema operativo.

En Shell script, una función es un bloque de código que contiene una secuencia de órdenes, instrucciones o comandos. Estas órdenes cumplen con una tarea específica de una aplicación que la invocó.

Ejercicios propuestos

1. Investigue el funcionamiento de **bc (basic calculator)** en línea de comandos, el cual es un programa Linux que permite operaciones matemáticas que pueden ser incluidos en scripts.

2. Desarrolle un programa en Shell script con las funcionalidades de una calculadora, que permita realizar las operaciones básicas, el programa debe solicitar dos números y la operación a realizar, luego mostrar el resultado.

3. Desarrolle un programa en Shell script que pida al usuario ingresar una serie de números (finaliza con un 0) y calcule el promedio de esos números.

4. Desarrolle un programa en Shell script que genere la tabla de multiplicar de un número, además de solicitar el rango en el que mostrará los resultados, todo esto siendo ingresado por el usuario.

5. Desarrolle un programa en Shell script que convierta temperaturas entre Celsius y Fahrenheit. El usuario debe ingresar la temperatura y la unidad de origen.

6. Desarrolle un programa en Shell script que calcule el factorial de un número ingresado por el usuario de forma iterativa.

7. Desarrolle un programa en Shell script que lea por teclado tres números enteros positivos y verifique cuál es el mayor.

8. Realice un programa en Shell script que calcule el área de un círculo según su radio, este debe de se ingresado por el usuario.

9. Desarrolle un programa en Shell script que calcule el área de un rectángulo a partir de su base y altura. El usuario debe ingresar por teclado ambos valores.

10. Desarrolle un programa en Shell script que calcule el área de un triángulo dado su base y altura. El usuario debe ingresar por teclado estos valores.

11. Implemente mediante Shell script un programa que calcule el área de un trapecio según su base y su altura.

12. Desarrolle un programa en Shell script que calcule los primeros n términos de la serie de Fibonacci. El usuario debe ingresar el valor de n. La serie de Fibonacci es una sucesión infinita de números enteros: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

13. Desarrolle un programa en Shell script que calcule la suma de los primeros n términos de la serie armónica. El usuario debe ingresar el valor de n.

14. Desarrolle un programa en Shell script que cuente el número de caracteres en una cadena ingresada por teclado por el usuario.

15. Desarrolle un programa en Shell script que liste todos los archivos y directorios en el directorio actual.

16. Realice un programa en Shell script que pida al usuario el nombre de un archivo guardado en el sistema y muestre cuántas líneas contiene.

17. Desarrolle un programa en Shell script que pida al usuario el nombre de un archivo y una palabra, y que muestre cuántas veces aparece la palabra en el archivo

18. Desarrolle un programa en Shell script que pida al usuario el nombre de un archivo y un nuevo nombre, y que copie el archivo original al nuevo nombre

19. Desarrolle un programa en Shell script que pida al usuario el nombre de un archivo y lo elimine. Utiliza el comando rm. Asegúrese de preguntar al usuario si está seguro de querer eliminar el archivo.

20. Desarrolle un programa en Shell script que muestre por pantalla la versión del sistema operativo, la fecha del sistema, el directorio actual y el usuario.

Sitios de interés

(I) CURSO 14146 de Shell scripts en Linux, propuesto por la Universidad de Sevilla: Curso de bash

(II) Funciones en Bash: atareo con Linux que es una página Web que muestra ejercicios relacionados con funciones en shell scripts.

Autoevaluación del Capítulo

1. **¿Qué comando se utiliza para cambiar de directorio en shell script?**
   - cd.
   - mv.
   - ls.
   - pwd.
2. **¿Cómo se declara una variable en bash?**
   - nombre=valor.
   - var nombre: valor.
   - declare nombre valor.
   - let nombre=valor.
3. **¿Cuál es el operador para concatenar cadenas en shell script?**
   - variable1 $ variable2.
   - variable1 + variable2.
   - variable1 & variable2.
   - variable1.concat(variable2).
4. **¿Qué hace el comando echo $¿?**
   - Muestra el código de salida del último comando.
   - Imprime el contenido de una variable en base al último archivo abierto.
   - Muestra el número de argumentos pasados.
   - Reinicia el terminal y muestra el consumo de recursos.
5. **¿Qué significa el símbolo # al inicio de una línea en un script?**
   - Es un comentario.
   - Define una variable global para todo el script.
   - Marca el inicio de un bucle for y debe de ser cerrado igualmente.
   - Define una función lambda.

6. **¿Qué se entiende por función en bash?**
   - nombre_funcion() comandos .
   - function nombre_funcion comandos .
   - def nombre_funcion: comandos.
   - func nombre_funcion() =>comandos.
7. **¿Cuál es la finalidad de las variables de entorno?**
   - Guardar información del sistema y del usuario.
   - Definir funciones que el usuario no recuerde y luego solo ser llamadas.
   - Permitir hacer cálculos aritméticos sin librerías externas.
   - Ejecutar scripts automáticamente, sin intervención manual.
8. **¿Qué comando se utiliza para listar archivos y directorios?**
   - ls.
   - dir.
   - list.
   - show.
9. **¿Cómo se puede leer la entrada del usuario en un script?**
   - read variable.
   - input variable.
   - scan variable.
   - get variable.
10. **¿Cuál es el acrónimo del interfaz portable estandard de Unix ?**
    - POSIX.
    - ISO/IEC 9945.
    - GNU Core Utilities.
    - Single UNIX Specification.

CAPÍTULO VI

Gestión de Archivos y Carpetas

Introducción

Tras aprender lo necesario de los capítulos anteriores, como el poder familiarizarse con el entorno de Linux, es importante que no solo se memorice para qué sirve cada comando o cómo realizar cierta función usando shell, sino que también se comprenda cómo se gestionan los archivos para crear, organizar, modificar y administrar los diferentes directorios en nuestra área de trabajo.

Además, es importante entender la importancia de controlar quién puede tener acceso a ciertos archivos, ya que algunos pueden ser solo de lectura para ciertos usuarios debido a la seguridad que se busque dar, editables para otros o directamente inaccesibles. Esto permite que el sistema sea seguro frente a usuarios que no busquen dar buen uso del mismo.

Un **sistema de archivos** es el componente principal que permite **estructurar y organizar** la escritura, búsqueda, lectura, almacenamiento, edición y eliminación de archivos en un dispositivo de memoria. Su objetivo es asegurar que el usuario pueda identificar los archivos sin errores y acceder a ellos lo más rápido posible. Además, los sistemas de archivos otorgan a los archivos características como convenciones para su nombramiento, atributos específicos y mecanismos de control de acceso. También funcionan como una interfaz crítica entre el sistema operativo y los dispositivos de almacenamiento conectados, tanto internos como externos, como las memorias USB.

En este capítulo, el lector profundizará en cómo Linux organiza y gestiona su sistema de archivos. La jerarquía de directorios en Linux es clave para mantener el orden y facilitar la búsqueda de información. Aprenderá a gestionar permisos de acceso, asegurando que los archivos estén protegidos y disponibles solo para quienes deben tener acceso a ellos.

Al finalizar este capítulo, el lector comprenderá la estructura y el funcionamiento de los sistemas de archivos en Linux. Aprenderá cómo organizar sus archivos y directorios de manera eficiente, cómo configurar permisos para garantizar la seguridad de la información, y cómo utilizar las herramientas necesarias para mantener el rendimiento y la integridad de su sistema de archivos.

Este capítulo también incluirá actividades prácticas, problemas resueltos y recursos adicionales que le ayudarán a afianzar su aprendizaje.

¡Bienvenidos!

Sistemas de Archivos, estructura y organización

Tipos de Archivos

Los sistemas de archivos en Linux manejan diferentes tipos de archivos, cada uno con características y propósitos específicos. A continuación, en la Tabla VI.1, se describen algunos de los tipos de archivos más comunes:

Tabla VI.1: Tipos de Archivos en Sistemas de Archivos Linux

Tipo de Archivo	Descripción
Archivos de Texto	Son archivos que contienen datos en un formato legible por humanos, como código fuente, scripts y documentación. Ejemplos de extensiones incluyen `.txt`, `.c`, `.py`.
Archivos Binarios	Son archivos que contienen diferentes tipos de información (datos ejecutables, imágenes, audio y video) codificados en binario. Ejemplos son `.exe`, `.png`, `.mp3`.
Archivos de Dispositivo	Representan dispositivos hardware en el sistema, se encuentran en el directorio `/dev` y permiten la interacción con discos duros, impresoras y terminales.
Archivos de Directorio	Actúan como contenedores para otros archivos, organizando los mismos en una estructura jerárquica.
Archivos de Enlace	Referencian a otros archivos en el sistema mediante enlaces simbólicos conocidos como ("symlinks") o por medio de enlaces duros, que apuntan al mismo inode.
Archivos Especiales	Incluyen archivos FIFO (First In First Out) y archivos de sockets, que permiten la comunicación entre procesos.

Comandos Linux para gestión de archivos y carpetas

La gestión de archivos en el sistema Linux se realiza a través de comandos que permiten crear, mover, copiar, renombrar y eliminar archivos y directorios. Acciones que se usarán en el día a día por su utilidad y relevancia, como:

- **ls (List):** Lista los archivos y carpetas del directorio actual y a su vez permite mostrar más detalles.

Comando para listar archivos y carpetas

```
ls
ls -l # Muestra detalles como permisos y tamaño
```

- **cd (Change Directory):** Cambia de directorio. Se utiliza para navegar entre carpetas.

Comando que permite el cambio entre directorios

```
cd /home/sysadmin/Docker
```

- **mkdir (Make Directory):** Crea una o varias carpetas nuevas.

Comando para crear carpetas en el directorio especificado

```
mkdir carpeta_nueva
```

- **touch (Create a new, empty file):** Crea un archivo vacío o reemplaza la fecha del archivo.

Comando para crear un archivo vacío nuevo

```
touch nuevo_archivo.txt
```

- **rm (Remove):** Elimina archivos o directorios de forma permanente.

Comando para eliminar archivos o carpetas

```
rm archivo.txt
rm -r carpeta/
```

- **cp (Copy):** Copiar de una ubicación a otra archivos o directorios.

Comando para copiar archivos/directorios

```
cp archivo.txt /ruta/nueva/
cp -r carpeta/ /ruta/nueva/
```

- **mv (Move):** Mover o renombrar a una nueva ubicación archivos o directorios.

Comando para mover o renombrar archivos

```
mv archivo.txt /ruta/destino/
mv archivo_viejo.txt archivo_nuevo.txt #
Renombra el archivo
```

Gestión de permisos de archivos y directorios

En Linux, la gestión de permisos es primordial para garantizar la seguridad y privacidad de los archivos y directorios. Los permisos determinan quién puede acceder, modificar o ejecutar archivos o carpetas en el sistema, estableciendo un control minucioso sobre los recursos.

Tipo de Archivo:

Los archivos pueden ser de varios tipos. A continuación, en la Tabla VI.2 se describen algunos de los más comunes:

Tabla VI.2: Tipos de Archivo en Linux

Identificador	Tipo de Archivo
-	Archivo regular (FILE)
d	Directorio (DIRECTORY)
l	Enlace simbólico (LINK)
s	Socket
b	Bloque (BLOCK)
P	tubería (PIPE)

Asignación de permisos con números

En Linux, los permisos se pueden asignar utilizando un sistema numérico llamado **notación octal**. Cada archivo o directorio tiene permisos que se asignan a tres tipos de usuarios: el **propietario**, el **grupo** al que pertenece el archivo, y **otros** usuarios. Estos permisos se representan mediante tres dígitos en la notación numérica.

Estructura de la notación numérica de permisos

La notación octal utiliza tres dígitos, donde cada uno tiene una función específica. La Tabla VI.3 muestra su estructura:

Tabla VI.3: Estructura de los permisos en notación numérica (octal)

Dígito	Qué representa
Primer dígito	Define los permisos del propietario del archivo.
Segundo dígito	Define los permisos del grupo al que pertenece el archivo.
Tercer dígito	Define los permisos de los demás usuarios (otros).

Cada uno de estos dígitos se forma a partir de la suma de valores numéricos asociados a tres tipos de permisos (véase Tabla VI.4): **lectura**, **escritura** y **ejecución**. Estos valores son los siguientes:

Tabla VI.4: Asignación de valores numéricos a permisos

Permiso	Descripción	Valor Numérico
r	Lectura (read)	4
w	Escritura (write)	2
x	Ejecución (execute)	1

Permisos y su representación binaria del 1 al 7

Los permisos en Linux también pueden representarse en binario. Cada tipo de permiso se corresponde con un número binario que ayuda a visualizar cómo se asignan los permisos. La Tabla VI.5 muestra las combinaciones de permisos correspondientes a los números del 1 al 7, donde se utiliza de representación binaria y notación de permisos en Linux (`rwx` - lectura, escritura, ejecución):

Tabla VI.5: Permisos y su representación binaria del 1 al 7

Número	Valor (4+2+1)	RWX	Descripción
1	1	001	Solo ejecución
2	2	010	Solo escritura
3	2+1	011	Escritura y ejecución
4	4	100	Solo lectura
5	4+1	101	Lectura y ejecución
6	4+2	110	Lectura y escritura
7	4+2+1	111	Lectura, escritura y ejecución

Asignación de permisos con símbolos

Además de la notación numérica, los permisos en Linux también asignan con símbolos. Esta notación simbólica modifica los permisos de forma granular para usuarios, grupos y otros. Los principales símbolos que se utilizan se muestran en la Tabla VI.6:

Tabla VI.6: Símbolos para la asignación de permisos

Símbolo	Descripción
u	Usuario (owner)
g	Grupo (group)
o	Otros (others)
a	Todos (all)
r	Lectura (read)
w	Escritura (write)
x	Ejecución (execute)
+	Añadir permiso
-	Quitar permiso
=	Establecer un permiso específico

Ejercicios resueltos de asignación de permisos

En el siguiente apartado, se muestran ejercicios resueltos de asignación de permisos, con el uso de notación numérica y simbólica.

1. **Ejercicios con notación numérica**

   a) Otorgue permisos de lectura y escritura al archivo `permisos.txt`:

   Comando para otorgar permisos de lectura y escritura

   ```
   chmod 666 permisos.txt
   ls -l permisos*
   ```

   b) Otorgue permisos solo de lectura a todos en el archivo `permisos2.txt`:

   Comando para otorgar solo permisos de lectura a todos

   ```
   chmod 444 permisos2.txt
   ls -l permisos*
   ```

   c) Asigne permisos de lectura y escritura al usuario, y permisos de ejecución al grupo y otros en el archivo `permisos3.txt`:

   Comando para otorgar permisos de lectura y escritura al usuario

   ```
   chmod 611 permisos3.txt
   ls -l permisos*
   ```

   d) Asigne permisos de escritura y ejecución (`w,x`) al usuario y grupo, y permisos de lectura y ejecución (`r,x`) a otros en el archivo `permisos4.txt`:

   Comando para otorgar permisos de w

   ```
   chmod 335 permisos4.txt
   ls -l permisos*
   ```

2. **Ejercicios con notación simbólica**

   a) Otorgue permisos solo de ejecución a todos los usuarios en el archivo `permisos12.txt`:

Comando para otorgar permisos de solo ejecución a todos los usuarios

```
chmod a+x permisos12.txt
ls -l permisos*
```

b) Otorgue permisos de lectura y escritura (`rw`) a todos en el archivo `permisos13.txt`:

Comando para otorgar permisos de lectura y escritura a todos

```
chmod a+rw permisos13.txt
ls -l permisos*
```

c) Asigne permisos de lectura y ejecución (`r,x`) al usuario y al grupo en el archivo `permisos14.txt`:

Comando para otorgar permisos de lectura y ejecución al usuario y grupo

```
chmod ug=rx permisos14.txt
ls -l permisos*
```

d) Asigne permisos de solo escritura (`w`) a todos en el archivo `permisos1.txt`:

Comando para otorgar solo permisos de escritura a todos

```
chmod a+w permisos1.txt
ls -l permisos*
```

Ejercicios propuestos de asignación de permisos

1. **Ejercicios con notación numérica**

a) Cree un archivo llamado `documentoimportante.txt` y otorgue permisos de escritura (`w`) al usuario, y permisos de lectura (`r`) al grupo y a otros.

b) Al archivo `proyectos.txt` otorgue permisos de lectura, escritura y ejecución (`rwx`) al usuario, permisos de lectura y ejecución (`rx`) al grupo, y permisos de lectura (`r`) a otros.

c) Cree un archivo llamado `documentoimportante2.txt` y otorgue permisos de solo lectura (`r`) a todos.

d) Al archivo `permisos45.txt` asigne permisos de escritura (`w`) al usuario y a otros, y permisos completos de lectura, escritura y ejecución (`rwx`) al grupo.

e) Cree un archivo nuevo y conceda permisos de escritura y ejecución (`wx`) al usuario y al grupo.

f) Al archivo llamado `permisos104.txt` ceda permisos de lectura y ejecución (`rx`) a todos los usuarios.

g) Explique el resultado de lo siguiente: `chmod 754 docs.txt`.

h) Asigne permisos de escritura (`w`) al usuario, y permisos de lectura (`r`) al grupo y a otros al documento `docgeneral.txt`.

i) De permisos de ejecución (`x`) al usuario y al grupo, y elimine todos los permisos a otros del archivo `scriptseguro.sh`.

j) ¿Qué hace el siguiente comando: `chmod 640 reporte.pdf`?.

2. **Ejercicios con notación simbólica**

a) Otorgue permisos de ejecución a todos los usuarios en el archivo `script12.sh`.

b) Otorgue permisos de lectura y escritura a todos los usuarios en el archivo `documento13.txt`.

c) Asigne permisos de lectura y ejecución al usuario y al grupo para el archivo `programa14`.

d) Otorgue permisos de escritura solo al grupo en el archivo `notas15.txt`.

e) Revoque los permisos de ejecución para otros en el directorio `proyecto16`.

f) Conceda permisos de lectura y escritura al usuario y al grupo en el archivo `datos17.dat`.

g) Elimine los permisos de escritura para otros en el archivo `configuracion18.cfg`.

h) Conceda permisos de ejecución al grupo en el archivo `tarea19.sh`.

i) Otorgue todos los permisos a todos los usuarios en el archivo `archivo20.txt`.

j) Revoque los permisos de lectura para otros en el directorio `documentos21`.

Comandos Linux para compresión y descompresión de archivos

La compresión de archivos en Linux reduce el tamaño de los archivos facilitando su transferencia, lo cual no solo es útil cuando no se tiene suficiente almacenamiento, sino que permite compartir diferentes documentos en un único archivo. A continuación, se presentan sus comandos:

Comprimir archivos

1. **Comando `tar`**: `tar` es muy utilizado para agrupar varios archivos en un único archivo comprimido. Aunque no realiza la compresión por sí solo, se puede combinar con otras herramientas para hacerlo.

   Comando para crear un archivo comprimido

   ```
   tar -cvf respaldo.tar /ruta/archivo/
   ```

   - `-c`: Crear el nuevo archivo tar.
   - `-v`: Muestra el progreso de la operación.
   - `-f`: Indica el nombre del archivo que se va a crear.

2. **Comando `gzip`**: `gzip` es una herramienta para comprimir archivos. Se utiliza a menudo junto con `tar` para comprimir los archivos empaquetados.

   Comando para comprimir un archivo con gzip

   ```
   gzip archivo.txt
   ```

   - Comprime el archivo `archivo.txt` y lo reemplaza por `archivo.txt.gz`.

3. **Comando `zip`**: `zip` se usa para comprimir uno o varios archivos en un único archivo comprimido con extensión `.zip`.

   Comando para crear un archivo .zip

   ```
   zip archivo.zip archivo1 archivo2
   ```

   - Comprime los archivos `archivo1` y `archivo2` en un único archivo `archivo.zip`.

Descomprimir archivos

1. **Comando `tar`**: Permite descomprimir archivos creados con el comando `tar`.

   Comando para descomprimir un archivo
   ```
   tar -xvzf example.tar.gz
   ```

2. **Comando `gunzip`**: Utilizado para descomprimir archivos con la extensión `.gz`.

   Comando para descomprimir un archivo .gz
   ```
   gunzip archivo.gz
   ```

   - Descomprime el archivo `archivo.gz` y lo restaura a su estado original.

3. **Comando `unzip`**: Permite descomprimir archivos `.zip`.

   Comando para descomprimir un archivo .zip
   ```
   unzip expample.zip
   ```

   - Descomprime el archivo `example.zip`.

Otras opciones

- Para listar el contenido de un archivo comprimido sin descomprimirlo:

  Comando para listar el contenido de un archivo .tar
  ```
  tar -tf archivo.tar
  ```

  - `-t`: Listar el contenido del archivo comprimido.
  - `-f`: Especificar el nombre del archivo.

- Para comprimir con `gzip`:

  Comando para comprimir con gzip
  ```
  gzip -9 archivo.txt
  ```

  - `-9`: Comprime al máximo.

Programación en C para leer, escribir, crear, copiar y borrar archivos en Linux

Como se ha visto en este capítulo, la gestión de archivos es sumamente importante para poder llevar una buena organización de estos, además de brindar seguridad y/o protección sobre el contenido a los usuarios destinados. Es por eso que para tareas más afines mediante lenguaje de C se aplicarán estos principios, ya que en un entorno profesional no siempre es suficiente usar la herramienta de línea de comandos como ´cat, cp, rm´; sino que también el conocer el proceso de automatización mediante operaciones condicionales para el manejo de grandes volúmenes de archivos.

Diferencias en el manejo de archivos entre la línea de comandos y el lenguaje C

La principal diferencia entre el manejo de archivos entre la línea de comandos y el lenguaje C, es la limitación propia de las tareas, por ejemplo, dentro de la línea de comandos se realizan tareas típicas como visualizar, copiar o eliminar el contenido. En cambio, cuando se requiere mayor granularidad, como leer un segmento, procesar línea a línea, actualizar segmentos del archivo, el lenguaje **C** proporciona un mejor manejo por el uso de librerías.

Para la correcta manipulación de archivos es indispensable considerar los **permisos de acceso** y los **modos de apertura**, conforme a lo expuesto en la Tabla VI.6. Entre los modos habituales se incluyen `r` (lectura), `w` (escritura con truncado), `a` (anexado), así como sus variantes de lectura y escritura `r+`, `w+` y `a+`.

A continuación, se presenta paso a paso la lectura, escritura, copia y eliminación de archivos, con el uso de la línea de comandos y programación en C. Se utilizará un archivo de texto base con el siguiente contenido:

Archivo de Ejemplo: archivo.txt

```
Tareas incompletas:
1. Universidad de las Fuerzas Armadas ESPE
2. I love Linux
3. Bienvenido al mundo de la Programación.
4. echo "Hola Mundo"
```

Nota: Para verificar que el archivo se haya guardado correctamente, se comprueba en la terminal con la instrucción: **cat archivo.txt**.

Lectura de Archivos

Para esta sección se emplean funciones como:

- **fgetc**: permite la lectura de un único carácter.
- **fgets**: posibilita la lectura de una línea completa.
- **fread**: realiza la lectura de bloques de datos.

Ejemplo de lectura de una línea de un archivo

El siguiente programa en C muestra la forma de leer únicamente la primera línea del archivo `archivo.txt`:

Creación del programa con extensión en C

```
nano leer_archivo.c
```

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("archivo.txt", "r");
    if (file == NULL) {
        perror("Error al abrir el archivo");
        return 1;
    }
    char buffer[100];
    if (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("Contenido: %s", buffer);
    }
    fclose(file);
    return 0;
}
```

Listing VI.1: Lectura de una única línea

El programa se compila y ejecuta en la terminal con los siguientes comandos:

Compilación del Programa

```
gcc leer_archivo.c -o leer_archivo
./leer_archivo
```

La ejecución del programa muestra en pantalla el contenido de la primera línea del archivo `archivo.txt`.

Escritura de Archivos

Para esta sección se utiliza la función:

- **fputs**: permite escribir el texto especificado en un archivo.

Ejemplo de escritura en un archivo de texto

El siguiente programa en C ejemplifica la creación de un archivo de texto y la escritura de contenido en él:

Creación del programa con extensión en C

```
nano crear_archivo.c
```

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("archivo2.txt", "w");
    if (file == NULL) {
        perror("Error al abrir el archivo");
        return 1;
    }
    fputs("Materias de Software:\n1. Programacion Web\n2. Base de Datos\n3. Sistemas Operativos\n", file);
    fclose(file);
    printf("Archivo creado y modificado con exito.\n");
    return 0;
}
```

Listing VI.2: Creación e ingreso de información en un archivo

El programa se compila y ejecuta en la terminal con los siguientes comandos:

Compilación del Programa

```
gcc crear_archivo.c -o crear_archivo
./crear_archivo
cat archivo2.txt
```

La ejecución mostrará en pantalla la confirmación de creación y modificación del archivo, y el comando `cat` permitirá visualizar su contenido.

Copiar Archivos

Para esta sección se emplean funciones como:

- **fopen**: utilizada para abrir el archivo original en modo de lectura (r").

- Modo **"w"**: si el archivo destino no existe, se crea; si ya existe, se sobrescribe.

Ejemplo de copia de un archivo

El siguiente programa en C ejemplifica cómo copiar el contenido del archivo `archivo.txt` a un nuevo archivo denominado `copia.txt`:

Creación del programa con extensión en C

```
nano copiar_archivo.c
```

```c
#include <stdio.h>

int main() {
    FILE *src = fopen("archivo.txt", "r");
    FILE *dest = fopen("copia.txt", "w");
    if (src == NULL || dest == NULL) {
        perror("Error al abrir los archivos");
        return 1;
    }
    char buffer[1024];
    size_t bytes;
    while ((bytes = fread(buffer, 1, sizeof(buffer), src)) > 0) {
        fwrite(buffer, 1, bytes, dest);
    }
    fclose(src);
    fclose(dest);
    printf("Archivo copiado con  xito .\n");
    return 0;
}
```

Listing VI.3: Copia de información entre archivos

El programa se compila y ejecuta en la terminal mediante los siguientes comandos:

Compilación del Programa

```
gcc copiar_archivo.c -o copiar_archivo
./copiar_archivo
cat copia.txt
```

La ejecución generará un archivo denominado `copia.txt`, cuyo contenido será idéntico al de `archivo.txt`.

Borrar Archivos

A continuación, se muestran funciones para borrar líneas en específico o archivos completos.

- **remove**: función empleada para eliminar un archivo completo del sistema.
- **fopen** y **fputs**: permiten abrir un archivo y reescribir su contenido para borrar líneas específicas.

Ejemplo de borrado de un archivo

El siguiente programa en C muestra cómo eliminar un archivo denominado `archivo.txt`:

Creación del programa con extensión en C

```
nano borrar_archivo.c
```

```c
#include <stdio.h>

int main() {
    const char *filename = "archivo.txt";
    if (remove(filename) == 0) {
        printf("Archivo '%s' borrado con  xito .\n", filename);
    } else {
        perror("Error al borrar el archivo");
    }
    return 0;
}
```

Listing VI.4: Borrado completo de un archivo

El programa se compila y ejecuta en la terminal con los siguientes comandos:

Compilación del Programa

```
gcc borrar_archivo.c -o borrar_archivo
./borrar_archivo
```

Ejemplo de borrado de líneas en un archivo

Cuando se requiere eliminar únicamente una línea específica de un archivo, el procedimiento consiste en leer el contenido, omitir la línea seleccionada y reescribirlo en un archivo temporal que luego reemplaza al original.

Creación del programa con extensión en C

```
nano borrar_linea.c
```

```c
#include <stdio.h>
#include <string.h>

int main() {
    const char *filename = "archivo.txt";
    const char *tempfile = "temp.txt";
    FILE *src = fopen(filename, "r");
    FILE *temp = fopen(tempfile, "w");

    if (src == NULL || temp == NULL) {
        perror("Error al abrir los archivos");
        return 1;
    }

    char buffer[1024];
    int line_to_delete = 3; // L nea que se desea borrar
    int current_line = 1;

    while (fgets(buffer, sizeof(buffer), src) != NULL) {
        if (current_line != line_to_delete) {
            fputs(buffer, temp);
        }
        current_line++;
    }

    fclose(src);
    fclose(temp);

    remove(filename); // Se elimina el archivo original
    rename(tempfile, filename); // El archivo temporal se renombra

    printf("L nea %d eliminada con  xito .\n", line_to_delete);
    return 0;
}
```

Listing VI.5: Eliminación de una línea específica en un archivo

El programa se compila y ejecuta en la terminal mediante:

Compilación del Programa

```
gcc borrar_linea.c -o borrar_linea
./borrar_linea
cat archivo.txt
```

Este procedimiento elimina la línea seleccionada y actualiza el archivo original sin intervención manual directa sobre su contenido.

Conclusiones del capítulo y lecciones aprendidas

Este capítulo ha estado enfocado en comprender los fundamentos esenciales para trabajar con archivos en el lenguaje de programación C, explorando desde la creación de archivos, su lectura, escritura, hasta la eliminación de contenidos y archivos completos. Se espera que se pueda mejorar en el uso de las funciones como `fopen`, `fwrite`, `fread`, `remove`, y `fputs`, las cuales forman parte de la biblioteca estándar de C (`stdio.h`), para la manipulación eficiente de datos almacenados en el sistema de archivos.

Los ejercicios desarrollados reforzaron las habilidades en la gestión de archivos, mediante el uso de archivos temporales, buffers de lectura y buenas prácticas.

Lecciones Aprendidas

- El manejo de archivos en C está profundamente vinculado a la biblioteca estándar `stdio.h`, que proporciona funciones como `fopen`, `fclose`, `fread`, `fwrite` y `remove`.
- Al manipular archivos hay que tener mucho cuidado con los errores, ya que se debe revisar que los punteros a los archivos no sean `NULL` antes de proceder con operaciones de lectura o escritura.
- La función `remove` es útil para la eliminación completa de archivos, mientras que la modificación de contenido requiere la creación de un archivo temporal y el uso de funciones como `fgets` y `fputs`.
- La manipulación de archivos debe ser realizada con cuidado para evitar problemas como fugas de memoria, archivos corruptos o pérdida de datos.
- Este capítulo ha seguido las bases teóricas y prácticas descritas en libros como *The C Programming Language* de Kernighan y Ritchie, y textos especializados como *Programming in C* de Stephen G. Kochan, los cuales son pilares fundamentales en el aprendizaje de este lenguaje.

Sitios de interés

(I) Gestión de archivos en Linux, artículo de la página Linux Console: Linux Console ofrece guías detalladas sobre cómo gestionar archivos desde la terminal de Linux, incluyendo ejemplos prácticos de comandos esenciales para personas que estan iniciando en el mundo de la informática.

(II) La guía definitiva de línea de comandos de Linux: FreeCodeCamp presenta un tutorial completo sobre Bash, incluyendo secciones específicas para manipular archivos en sistemas Linux, dando un conocimiento básico e intermedio para el manejo adecuando de estos.

(III) 60 comandos básicos de Linux que todo usuario debe saber: Hostinger presenta una lista de comandos para la gestión de archivos, directorios y procesos en Linux.

(IV) Gestión de archivos y directorios en Linux: 4Geeks ofrece una visión completa sobre cómo realizar una buena navegación, manipulación y configuración de permisos en archivos Linux.

Autoevaluación del Capítulo

1. **¿Qué comando permite listar archivos y directorios en Linux?**
   - ls.
   - cd.
   - touch.
   - rm.
2. **¿Qué opción del comando `chmod` permite otorgar permisos de lectura, escritura y ejecución al propietario?**
   - 7.
   - 5.
   - 3.
   - 4.
3. **¿Cuál de las siguientes funciones se utiliza para escribir contenido en un archivo en C?**

- fputs.
- fgets.
- fgetc.
- remove.

4. **¿Qué comando en Linux permite cambiar los permisos de un archivo?**
   - chmod.
   - ls.
   - rm.
   - mkdir.
5. **¿Qué número en notación octal representa permisos de solo lectura y ejecución?**
   - 5.
   - 6.
   - 4.
   - 3.
6. **¿Qué comando se utiliza para descomprimir un archivo `.tar.gz`?**
   - tar -xvzf.
   - gzip.
   - unzip.
   - gunzip.
7. **¿Cuál de las siguientes funciones de C permite leer una línea completa de un archivo?**
   - fgets.
   - fgetc.
   - fwrite.
   - remove.
8. **¿Qué símbolo en `chmod` permite revocar un permiso?**
   - -.

- +.
- =.
- r.

9. **¿Cuál es el valor numérico de los permisos de solo escritura en notación octal?**
   - 2.
   - 4.
   - 1.
   - 3.
10. **¿Qué función en C se utiliza para borrar un archivo completo?**
   - remove.
   - fputs.
   - fgets.
   - rename.

CAPÍTULO VII

Gestión de Procesos

Introducción

El concepto de ***proceso*** constituye uno de los pilares fundamentales de los sistemas operativos modernos. Un proceso puede entenderse como la abstracción de un programa en ejecución, a través de la cual se gestiona la interacción entre el usuario, las aplicaciones y los recursos del hardware [127].

La **gestión de procesos** comprende el conjunto de mecanismos y políticas que permiten al sistema operativo controlar la creación, ejecución, planificación, sincronización y finalización de procesos, optimizando el uso del procesador y garantizando tiempos de respuesta adecuados [124]. Esta gestión implica además la asignación y liberación de recursos críticos como CPU, memoria y dispositivos de entrada y salida.

Como se mencionó en el Capítulo I, el sistema operativo es un intermediario entre el hardware y las aplicaciones, este se encarga de distribuir los recursos entre los procesos en ejecución [31]. Por tal motivo, entender el ciclo de vida de un proceso y sus estados sirve para analizar el comportamiento y desempeño del sistema.

Este capítulo tiene como objetivo presentar los fundamentos de la gestión de procesos y su aplicación práctica. Al finalizar, el lector comprenderá qué es un proceso, sus estados, transiciones, estructura interna y el funcionamiento del bloque de control de procesos.

Parte del capítulo será el introducir comandos de Linux para la gestión de procesos, ejecución en primer y segundo plano y el análisis del rendimiento del procesador. El capítulo incluye además la programación de procesos en lenguaje C ANSI, permitiendo interactuar directamente con el sistema operativo y comprender su comportamiento a bajo nivel.

Finalmente, se proponen actividades prácticas y recursos complementarios que facilitan el aprendizaje, utilizando como plataforma de experimentación la interfaz de línea de comandos de Linux para escritorio y servidor, junto con el lenguaje C desarrollado por Dennis Ritchie [48].

May the Linux force be with you.

¡Bienvenidos!

Conceptos básicos

Proceso

Es **proceso** es un programa en ejecución. Para entenderlo, es necesario establecer qué es un programa de computadora. Un **programa** es un archivo que contiene instrucciones y datos almacenados en una memoria externa o secundaria como un disco duro. Por tanto, contiene el código que se ejecutará. No obstante, no está activo ni en ejecución siempre. En cambio, un proceso, es la instancia de ejecución de un programa en memoria, que incluye el estado de ejecución, recursos asignados, y es gestionado dinámicamente por el sistema operativo [31].

Un programa no consume recursos del sistema como CPU o memoria cuando no está en ejecución. En cambio, un proceso consume activamente CPU aun cuando no está en ejecución. Es decir, sigue ocupando algunos recursos del sistema, como memoria y espacio en la tabla de procesos, incluso cuando está en estado de espera o listo para ejecutarse. En resumen, un programa es el código estático almacenado en disco, mientras que un proceso es ese programa en ejecución, ocupando recursos del sistema como memoria y CPU [124].

Cada proceso que se inicia es identificado con un número de identificación único conocido como Process ID (PID), que es siempre un número entero y que es asignado por el kernel del sistema operativo, en el momento en que el proceso es creado [127].

Tipos de procesos

El identificar el tipo de proceso posibilita al sistema operativo aplicar políticas de administración y planificación para la optimización de recursos, pues cada tipo tiene diversos requisitos de recursos computacionales. Además, permite la priorización y planificación, la gestión de procesos especiales, la estabilidad del sistema, la gestión de la seguridad y control de acceso y la programación de tareas y automatización. A continuación, se explican brevemente algunos tipos de procesos reconocidos por el sistema operativo:

Procesos de Usuario: Son aquellos creados por el usuario, ya sea directamente desde la interfaz de línea de comandos, una Shell, o mediante la ejecución de una aplicación. Pueden ser aplicaciones, scripts, artefactos de software o cualquier tarea que un usuario necesita emplear [116].

Procesos del Sistema o del Kernel: Son procesos iniciados por el kernel y

gestionan la memoria, el almacenamiento y la red. Normalmente, los procesos del sistema tienen identificadores de proceso (PIDs) bajos y no suelen ser manipulados directamente por el usuario [127].

Demonios (Daemons): Son los procesos que se ejecutan en segundo plano y estos suelen comenzar cuando el sistema arranca. Por lo general, tienen nombres que terminan en "d". Estos procesos realizan tareas recurrentes o de espera, como escuchar conexiones de red, monitorear servicios o programar tareas.

Procesos Zombis: son aquellos que han terminado su ejecución, sin embargo, su entrada en la tabla de procesos no se ha eliminado aún.

Estado de un Proceso

A medida que el sistema operativo ejecuta un proceso, este va cambiando de estado. El estado de un proceso se define según la actividad actual de dicho proceso. Conocer el estado actual de un proceso es una forma de gestionar los recursos computacionales de manera eficiente por parte del sistema operativo. Cada proceso puede estar en uno de los estados siguientes:

- Nuevo. El proceso está siendo creado. Los procesos en este estado están cargados en la memoria, sin embargo, aún el sistema operativo no asigna recursos de CPU.

- Preparado. El proceso está a la espera de que le asignen a un procesador. Se trata de procesos que están cargados y listos en la cola de procesos para pasar al estado de ejecución

- En ejecución. Se están ejecutando las instrucciones por parte del CPU. Este es el único estado en el que el proceso ha sido asignado CPU por parte del sistema Operativo.

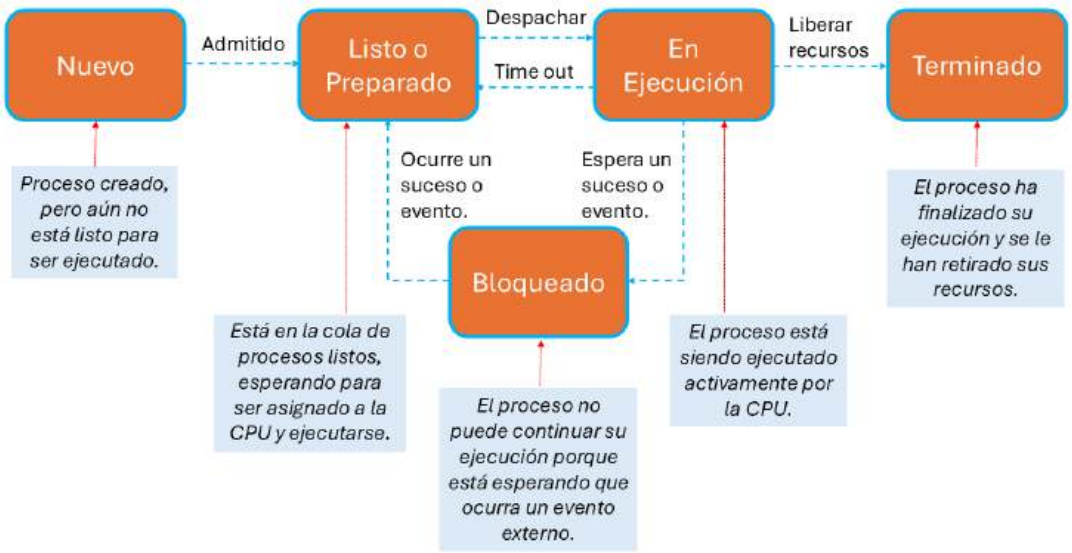
- Bloqueado. El proceso está esperando a que se produzca un suceso, es decir, necesita algún recurso que no ha sido asignado durante su ejecución, y solo continuará hasta que este recurso esté disponible o el evento ocurra. Por ejemplo, la terminación de una operación de E/S, la recepción de una señal o el acceso a un archivo.

- Terminado. Ha terminado la ejecución del proceso, por lo tanto, el sistema operativo ha liberado los recursos que le había asignado, como RAM, archivos abiertos y otros recursos de sistema.

Es importante señalar que, conocer en qué estado está cada proceso, sea nuevo, listo, en ejecución, bloqueado, o terminado, el sistema operativo puede planificar cuándo asignar el tiempo de CPU y otros recursos [31]. Así mismo, los estados de un proceso ayudan a gestionar la concurrencia y la sincronización, lo que permite evitar conflictos en el acceso a los recursos compartidos. Esto evita la sobrecarga y en consecuencia, mejora el desempeño general del sistema [124].

Cabe señalar que sólo puede haber un proceso ejecutándose en cualquier procesador en cada instante en concreto [127]. Sin embargo, pueden existir varios procesos preparados y en estado de espera en la cola de procesos. En la Figura VII.1 se muestra el diagrama de estados de un proceso.

Figura VII.1: Estados de un proceso.



Transiciones entre los estados de un proceso

Son cambios en el estado de un proceso generados por el sistema operativo, a medida que avanza su ciclo de vida. Estos cambios posibilitan al sistema operativo gestionan eficientemente los recursos y priorizar la ejecución de procesos. Las transiciones ocurren en respuesta a algunos eventos tales como interrupciones, disponibilidad de recursos o finalización de tareas. Las principales transiciones entre estados son las siguientes:

- Ninguno a nuevo: se crea desde cero un nuevo proceso para poder ejecutar el programa

- Nuevo a preparado: el proceso ha sido creado y está preparado para ser programado por el sistema operativo si dispone de recursos para ello.
- Preparado a ejecución: el sistema selecciona un proceso preparado y le asigna la CPU.
- Preparado a ejecución: el sistema selecciona un proceso preparado y le asigna la CPU.
- Ejecución a preparado: termina el tiempo de CPU del proceso, este vuelve al estado preparado, es decir, interrumpe para atender a otro de mayor prioridad.
- Ejecución a bloqueado: el proceso solicita un recurso o debe esperar un evento.
- Bloqueado a preparado: el evento ocurre o el recurso solicitado ya está disponible.
- Ejecución a terminado: el proceso finaliza y el sistema libera sus recursos.

Bloque de Control de Procesos

Cada proceso creado por la CPU tiene su propia información y la almacena en un registro conocido como un bloque de control de proceso (Process Control Block, PCB), también llamado bloque de control de tarea [116]. Un BCP contiene algunos atributos de información asociados con un proceso específico, entre los que se incluyen (véase Figura VII.2):

Identificador del Proceso: asignado para rastrear y diferenciar cada proceso para la asignación de recursos, planificación y comunicación entre procesos.

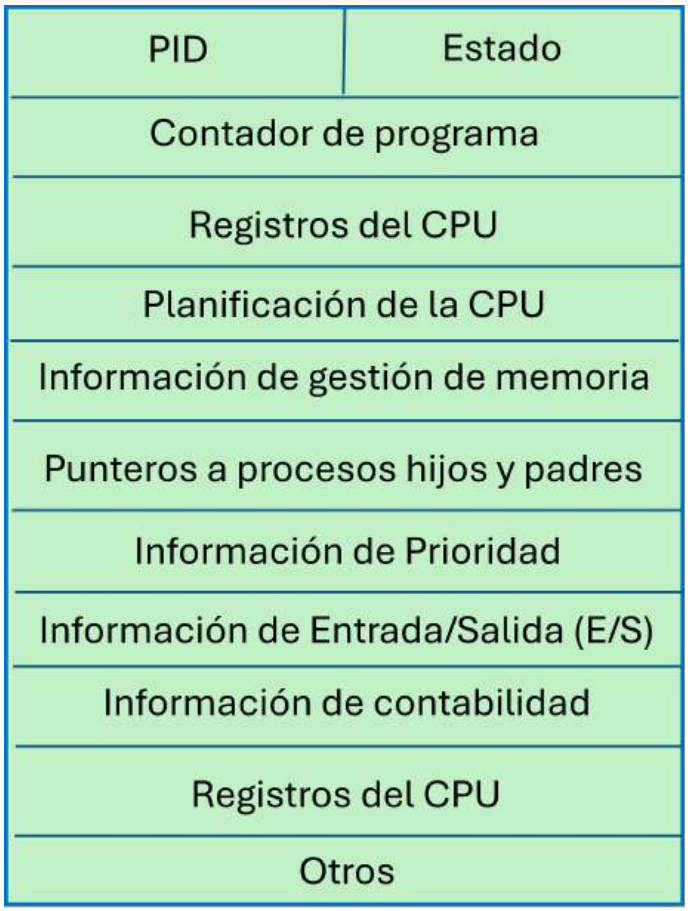
Estado del proceso: Nuevo, preparado, en ejecución, en espera, y terminado.

Contador de programa: Dentro del proceso es la siguiente instrucción que se va a ejecutar.

Registros de la CPU: Los registros varían en cuanto a número y tipo, dependiendo de la arquitectura de la computadora. Incluyen los acumuladores, registros de índice, punteros de pila y registros de propósito general, además de toda la información de los indicadores de estado.

Información de planificación de la CPU: La prioridad del proceso y los datos usados para su planificación (punteros a las colas de planificación, entre otros parámetros).

Figura VII.2: Bloque de Control de Proceso.



Información de gestión de memoria: Incluye información acerca del valor de los registros base y límite, las tablas de páginas, o las tablas de segmentos, dependiendo del mecanismo de gestión, de memoria utilizado por el sistema operativo.

Punteros a Procesos Hijos y Padres: Permite gestionar la relación jerárquica entre procesos, rastrear y organizar los procesos en una estructura de árbol, para la gestión de recursos, sincronización y finalización de procesos.

Información de prioridad: Prioridad del proceso, ayuda a planificador cuál de ejecuta primero.

Información de entrada/salida (E/S): Muestra los dispositivos y archivos que se usan por el proceso para operaciones de entrada y salida.

Información de contabilidad: Registra el consumo de recursos, es decir, el tiempo de CPU usado y otros datos.

Comandos Linux para gestión de Procesos

A continuación, se describen algunos de los comandos más utilizados en la gestión de procesos en la terminal de Linux:

- **cat /proc/cpuinfo**: Muestra la información del CPU.

  Comando donde se muestra la información del procesador

  ```
  cat /proc/cpuinfo
  ```

- **ps (process status):** Muestra los procesos en ejecución al instante.

  Comando para visualizar información resumida de procesos en ejecución

  ```
  ps
  ```

- **ps -aux:** Despliega todos los procesos en el sistema, incluyendo la información relacionada el propietario del proceso, el consumo de CPU, memoria, y el tiempo de ejecución, entre otros.

  Comando que lista los procesos en ejecución

  ```
  ps -aux
  ```

- **ps -ef:** Despliega una lista completa de procesos en formato estándar, mostrando información similar a ps -aux

  Comando que lista todos los procesos en ejecución con información completa

  ```
  ps -ef
  ```

- **ps -axjf**: Muestra el árbol jerárquico de los procesos.

  Árbol de procesos del sistema

  ```
  ps -axjf
  ```

- **pstree**: Muestra los procesos en ejecución con una estructura de árbol.

  Comando para visualizar el árbol de procesos en ejecución

  ```
  pstree
  ```

- **pstree -L #**: Visualización de los procesos en forma de árbol hasta el nivel que se indica.

Comando visualización el árbol de procesos en ejecución por capas

```
pstree -L#
```

- **ps aux | grep bash:** Permite visualizar los procesos en ejecución filtrando los procesos pertenecientes a bash.

Comando para visualizar el árbol de procesos exclusivos del bash

```
ps aux | grep bash
```

- **top:** Permite visualizar en tiempo real los procesos en ejecución incluyendo el consumo de recursos como CPU y memoria. Top es una herramienta con una interfaz visual de tipo cacter (string) útil para monitorear el rendimiento del sistema y para gestionar sus recursos

Comando que informa en tiempo real los procesos en ejecución

```
top
```

- **top –d 7:** Permite visualizar en tiempo real los procesos en ejecución actualizando dicha información cada 7 segundos.

Comando que muestra los procesos en ejecución actualizándola al transcurrir 7 segundos

```
top -d 7
```

- **top –u user:** Permite visualizar en tiempo real los procesos en ejecución de un determinado usuario.

Comando que muestra los procesos en ejecución del usuario especificado

```
top -u user
```

- **sleep:** Permite bloquear o suspender temporalmente la ejecución actual del procesador por un tiempo determinado.

Comando Linux que suspende la ejecución del procesador durante 5 segundos

```
sleep -5
```

- **kill - PID (identificdor del proceso):** Es una señal de cierre inmediato, que elimina el proceso y limpia su información almacenada en memoria.

Comando que elimina de la memoria el proceso especificado

```
kill -9 PID
```

- **killall program name:** Permite enviar señales a todos los proceses que forman parte del mismo programa para su cierre inmediato y limpiar su información almacenada en memoria.

Comando que elimina de la memoria todos los procesos especificados

```
killall gedit
```

Procesos en Primer y Segundo Plano

Procesos en Primer Plano

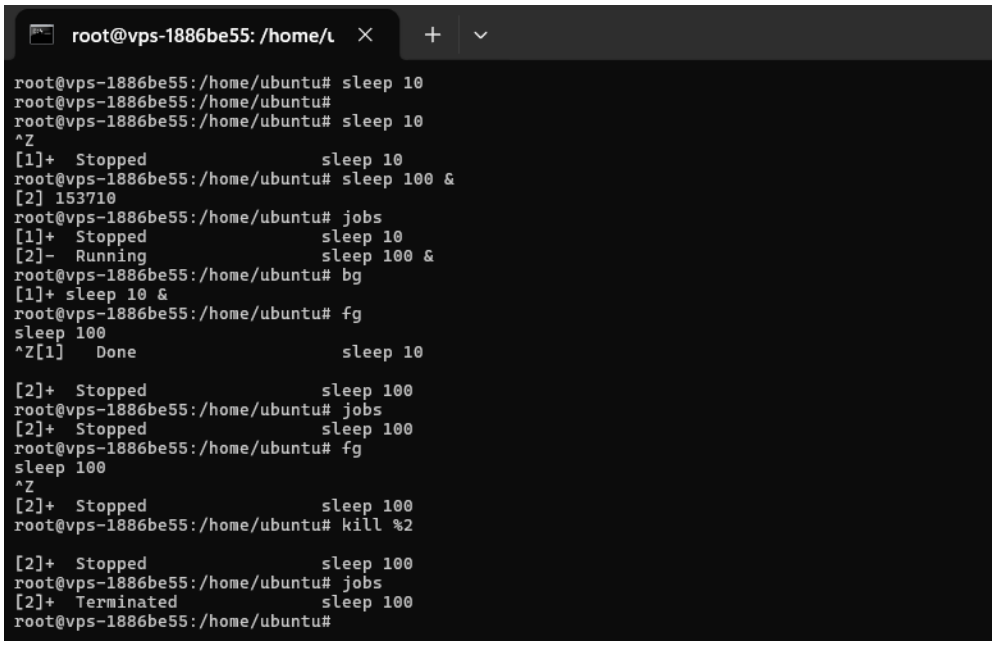
Son aquellos que requieren que un usuario los inicie o interactúe con ellos. Los programas y comandos se ejecutan como procesos en primer plano por defecto. Como es lógico, un programa en primer plano lanzado desde una interfaz de línea de comandos monopoliza dicha terminal, por lo que en principio no se puede ejecutar ningún otro programa simultáneamente. Como consecuencia, es necesario esperar a que finalice su ejecución antes de poder lanzar otro comando. Los procesos en primer plano se pueden suspender y llevar a segundo plano.

Procesos en Segundo Plano

Son aquellos que se ejecutan independientemente de un usuario. Fueron enviados a ejecución en un segundo plano, pues requieren que sean ejecutados conforme a algún requerimiento, como sincronización de datos, ubicación, actualización de información, etc. Por lo tanto, una vez

iniciada la ejecución de un programa en segundo plano, se deja libre la terminal desde el que se lanzó y la interfaz de línea de comandos muestra nuevamente el mensaje de aviso de espera de una nueva instrucción. La Figura VII.3 muestra este procedimiento.

Figura VII.3: Ejemplo en la consola de comandos de Linux de procesos en primer a segundo plano.



Conclusiones y lecciones aprendidas

Este capítulo ha sido enfocado en comprender los fundamentos básicos de la gestión de procesos en un sistema Operativo. Los lectores han aprendido que se trata de un programa en ejecución y de la asignación o liberación de recursos por parte del SO para este fin. Se ha explicado los tipos de procesos. Se ha ilustrado los estados y las transiciones de un modelo de procesos de cinco estados. Se ha puesto énfasis en el bloque de control de procesos como arquitectura de datos que contiene la información de cada proceso. Luego se vincularon estos conocimientos a la gestión de procesos en Linux y se presentaron varios comandos Linux para gestión de procesos. También se abordó el manejo de procesos en primer y segundo plano mediante ejemplos prácticos. La práctica con comandos de Linux permitió consolidar estos conceptos y mejorar la comprensión sobre la gestión de procesos.

Lecciones aprendidas

Conocer el estado actual de un proceso es una forma de gestionar los recursos computacionales de manera eficiente por parte del sistema operativo.
Un programa no consume recursos del sistema como CPU o memoria cuando no está en ejecución. En cambio, un proceso consume activamente CPU aun cuando no está en ejecución. Es decir, sigue ocupando algunos recursos del sistema, como memoria y espacio en la tabla de procesos, incluso cuando está en estado de espera o listo para ejecutarse.
Los procesos conocidos como demonios son procesos especiales que se ejecutan en segundo plano y suelen comenzar cuando el sistema arranca.
Los Procesos en primer plano son aquellos que requieren que un usuario los inicie o interactúe con ellos.
Los procesos en segundo plano se ejecutan sin interacción del usuario. Fueron enviados a ejecución en un segundo plano, pues requieren que sean ejecutados conforme a algún requerimiento, como sincronización de datos, ubicación, actualización de información, etc.

Sitios de interés

(I) Sistemas Operativos, Gestión de Procesos: José Luis Elvira: Video de procesos en SOs

(II) Introducción a la Gestión de Procesos: NanoEdun Video de gestión de procesos en Windows

(III) Introducción a la Gestión de Procesos. Lección en PDF escrita por Jesús Carretero Pérez, et al.

Autoevaluación del Capítulo

1. **¿Con cuál comando se lista los procesos activos en Linux?**
   - ps.
   - top.
   - kill.

- chmod.

2. **¿Qué es un proceso en el contexto de un sistema operativo?**
   - Una porción de memoria reservada para el sistema.
   - Un archivo ejecutable en el disco.
   - Un programa en ejecución.
   - Un segmento de código.
3. **¿Qué estado describe a un proceso que está esperando recursos del sistema?**
   - Running.
   - Waiting.
   - Ready.
   - Terminated.
4. **¿Qué comando en Linux permite finalizar un proceso?**
   - ps.
   - kill.
   - fg.
   - bg.
5. **¿Qué información proporciona el comando `ps`?**
   - Una lista de procesos activos y sus atributos.
   - La lista de archivos abiertos.
   - Los usuarios conectados al sistema.
   - El consumo de memoria de cada usuario.
6. **¿Qué es un hilo en un sistema operativo?**
   - La unidad básica de ejecución de un proceso.
   - Un programa independiente en ejecución.
   - Una estructura para manejar la memoria.
   - Un archivo del sistema.
7. **¿Qué función en C se usa para crear un nuevo proceso?**
   - fork().
   - exec().

- pthread$_c reate().wait()$.

8. **¿Qué hace la función `exec()` en C?**

- Detiene un proceso.
- Reemplaza el espacio de memoria del proceso con un nuevo programa.
- Crea un nuevo hilo.
- Libera recursos de un proceso.

9. **¿Qué diferencia a un proceso de un hilo?**
   - Un hilo comparte recursos del proceso principal, mientras que un proceso tiene sus propios recursos.
   - Un hilo es más lento que un proceso.
   - Un proceso siempre se ejecuta en segundo plano.
   - Los hilos no pueden ejecutarse simultáneamente.
10. **¿Qué recurso NO comparten los hilos de un mismo proceso?**
    - Espacio de memoria.
    - Recursos de E/S.
    - El contador de programa.
    - Variables globales.
11. **¿Qué función en C permite esperar a que un proceso hijo termine su ejecución?**
    - exec().
    - fork().
    - wait().
    - kill().
12. **¿Qué comando en Linux muestra la jerarquía de procesos en forma de árbol?**
    - top.
    - pstree.
    - ps.
    - lsof.
13. **¿Qué componente del sistema operativo se encarga de la planificación de procesos?**
    - Sistema de archivos.
    - El planificador (scheduler).

- El núcleo (kernel).
- El intérprete de comandos.

14. **¿Qué significa un proceso en estado "zombie¿**
    - Un proceso en ejecución sin interrupciones.
    - Un proceso bloqueado.
    - Un proceso que ha terminado, pero cuyo padre no ha leído su estado de salida.
    - Un proceso que no puede ser terminado.
15. **¿Qué comando se utiliza para reanudar un proceso en segundo plano en Linux?**
    - fg.
    - bg.
    - jobs.
    - stop.
16. **¿Qué es un “deadlock” en gestión de procesos?**
    - Una condición donde dos procesos comparten recursos eficientemente.
    - Un proceso que se ejecuta indefinidamente.
    - Una situación en la que dos o más procesos esperan indefinidamente por recursos bloqueados.
    - Un proceso que consume recursos continuamente.

CAPÍTULO VIII

Programación de Procesos en C para Linux

Introducción del Capítulo

Este capítulo tiene como objetivo capacitar al lector para que pueda programar procesos en Linux en lenguaje C. Con lenguaje C se pueden crear, gestionar y sincronizar procesos utilizando funciones del sistema operativo y bibliotecas específicas. De esa forma, el lector, observará y evidenciará como el sistema operativo los procesa y ejecuta.

Cabe señalar que este capítulo es relevante porque el kernel de Linux fue implementado con aproximadamente 99 % en C ANSI [76], [72] y por tanto su aplicación en esta plataforma, beneficiará a los lectores en el desarrollo de programas.

Como es costumbre en principio se otorgarán los aspectos teóricos en el contexto técnico, en el cual se abordarán tales temas como: Programación de Procesos en Linux, Gcc GNU C Compiler Collection, estructura de un programa en C, cómo compilar un programa en C, biblioteca de procesos POSIX, funciones para programación de procesos, creación de procesos en C, Función fork. Para fortalecer el aprendizaje, este capítulo cuenta con ejercicios resueltos y propuestos.

En este capítulo el lector también encontrará actividades de aprendizaje, programas propuestos, resueltos, lecciones aprendidas y sitios de interés que facilitarán la apropiación de su aprendizaje.

Como plataforma de entrenamiento que permitirá realizar las pruebas de concepto y formalizar el aprendizaje, se utilizará la interfaz de línea de comandos de una distribución de Linux tanto para escritorio como para servidor, así como el Lenguaje C de Dennis Ritchie [48].

Controla tus procesos, controla tu sistema.
¡Bienvenidos!

Fundamentos

Definición de procesos

Un proceso es un programa en ejecución. Un proceso es una entidad que todo sistema operativo utiliza para controlar el uso de los recursos computacionales. Como se ha señalado en ocasiones anteriores, el sistema operativo es un conjunto de programas que gestionan el hardware y permite que funcionen todas las aplicaciones de software que son cargados en la RAM para su ejecución. Dichas aplicaciones son procesos en ejecución. Por tanto, el concepto más importante y el centro de cualquier sistema operativo es el de proceso.

Un proceso en Linux cuando está en estado de ejecución en la memoria se constituye de las siguientes secciones: Las instrucciones del proceso (código); las variables globales del proceso (data), la memoria dinámica que genera el proceso (heap) y la preservación del estado en la invocación anidada de funciones (stack o pila).

Además, existen varios elementos que incorpora un proceso como la prioridad de ejecución que le indica al Sistema Operativo cuándo y cuántos recursos utilizar y el tiempo máximo de ejecución. Asimismo, está formado por un conjunto de recursos como archivos abiertos, datos internos del núcleo, un espacio de direcciones, uno o más hilos de ejecución y una sección de datos que contiene variables globales y locales.

Tipos de procesos

Proceso Padre

En sistemas operativos, es necesario considerar la jerarquía de los procesos que se están ejecutando. Un proceso iniciado por una aplicación de software o un comando es un proceso padre. Un proceso padre es un proceso que crea uno o más procesos hijos. Un proceso hijo es un proceso nuevo creado por un proceso padre y hereda ciertos atributos de su padre. Un proceso padre puede tener varios procesos hijo. Sin embargo, un proceso hijo sólo puede tener un padre. El sistema operativo asigna un número de identificación de proceso (número PID) al proceso hijo, y un número PPID al proceso padre cuando se inicia.

Proceso Hijo

Es un proceso creado por otro proceso durante su ejecución. El proceso que lo creó es un proceso padre. Los procesos hijo a veces son creados para correr un ejecutable desde un proceso existente. En Linux o Unix los procesos hijos se crean con la llamada a la función (). Los procesos regularmente son creados a través de un shell o terminal. En este caso la shell se convierte en proceso padre y el proceso ejecutado se convierte en hijo. En sistemas tipo Unix/Linux cada proceso tiene un padre excepto el proceso init.

Procesos Demonios

Un demonio es un programa que se ejecuta en segundo plano, fuera del control interactivo de los usuarios del sistema, porque carecen de interfaz o shell. El término demonio se usa fundamentalmente en sistemas GNU/Linux o Mac OS X. Estos se ejecutan con permisos de administrador (root) y usualmente proveen servicios. El no tener shell asociado se logra separando el proceso del shell, creando un proceso nuevo y terminando el proceso padre (el shell que lo inició).

Procesos Huérfanos

Son aquellos procesos en la que sus procesos padres son eliminados (i.e., killed). En otras palabras, los procesos huérfanos se crean cuando un proceso padre finaliza antes que sus procesos hijos. En tal caso, el proceso hijo queda huérfano y entonces es controlado por el proceso init . El proceso init es el responsable de inicializar el resto del sistema que no están en el Kernel.

Procesos Zombie

Son aquellos procesos hijos que han terminado su ejecución pero que, sin embargo, aún tiene una entrada en la tabla de procesos, hasta que el proceso padre obtenga la información del estado del proceso hijo que ha terminado. Hasta entonces el proceso terminado entra en estado zombie y ocupa aún los recursos del sistema. Cuando un proceso es terminado todos los recursos asociados con dicho proceso son liberados para que puedan ser usados por otros procesos.

Programación de Procesos en Linux

La programación de procesos en Linux implica la creación y gestión de procesos, que son instancias en ejecución de programas. Linux permite la ejecución simultánea de múltiples procesos. La comunicación y sincronización entre ellos puede lograrse mediante mecanismos como tuberías (pipes) o señales. Esta capacidad es esencial para el desarrollo de aplicaciones multitarea y sistemas concurrentes.

Gcc GNU C Compiler Collection

GCC es un conjunto de compiladores desarrollado por el Proyecto GNU [126]. En el contexto de la programación en Linux con C ANSI, GCC es la herramienta principal para compilar el código fuente escrito en C en un programa ejecutable. GCC facilita la traducción del código fuente a lenguaje de máquina, asegurando la portabilidad y optimización del código para el sistema operativo Linux [126].

Estructura de un programa en C

El lenguaje de programación C fue desarrollado por Dennis Ritchie en 1972 [74]. Actualmente es utilizado en Linux, en razón de que el Kernel fue creado con este lenguaje y debido a su eficiencia y capacidad para interactuar directamente con el hardware y los recursos del sistema [76], [72]. C ofrece un control a nivel de sistema que lo hace ideal para el desarrollo de aplicaciones de bajo nivel y sistemas operativos. Los programas escritos en C pueden utilizar funciones específicas del sistema operativo para crear, gestionar y comunicarse entre procesos, lo que es esencial en la programación de sistemas en entornos Linux.

Descripción: Este programa muestra la estructura de un programa en C para Linux.

```c
//Programa: Hello World
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Hello World, nice to meet you \n");
   system("sleep 5");
   return 0;
}
```

```
10 //I love Linux
```

Listing VIII.1: Programa: Hello World en C para Linux

Cómo compilar y ejecutar un programa en C

- **Forma de Compilación en C para Linux:** Una vez que se haya editado el programa en C, y no existen errores (warnings) de sintaxis o de semántica, se procede a compilar el programa en C, mediante el siguiente comando:

  Comando que inicia el proceso de transformación del programa en código fuente en un archivo binario ejecutable

  ```
  $ gcc -o programa programa.c
  ```

- **Forma de ejecución de un programa en C para Linux):** Si no existen errores en tiempo de compilación, en la interfaz de Linux se puede visualizar el resultado de la ejecución del programa.

  Comando para ejecutar un programa en C para Linux libre de errores

  ```
  $ ./programa
  ```

Creación de proceso en C

Los procesos hacen referencia a la ejecución activa de un programa. Los gestiona el kernel de Linux. En los sistemas Linux, los procesos suelen crearse a partir de un proceso existente conocido como proceso principal o padre. Los procesos son el centro del sistema operativo Linux. Los procesos son programas que se ejecutan en el host de Linux y que realizan operaciones como escribir en un espacio de memoria del disco, eliminar un archivo o ejecutar un servidor de correo electrónico.

A continuación, se realiza un código de ejemplo en el que se utiliza la función *system()* del lenguaje C. La función *system()* realiza la ejecución de comandos Linux en la consola desde un programa en C:

Función system()

Descripción: Este programa muestra la ejecución de comandos del Sistema.

```
1  //Program: The system Function in Linux - R
2  #include <stdio.h>
3  #include <stdlib.h>
4  int main()
5  {
6    printf("Imprime la version del Kernel en Linux \n");
7    system("uname -v");
8    printf("Listar 5 archivos \n");
9    system("ls -lh | head -5");
10   printf("Suspende el procesador 5 segundos \n");
11   system("sleep 5");
12   return 0;
13 }
14 //I love Linux
```

Listing VIII.2: Programa: Ejecución de comandos Linux mediante la función system de C para Linux

Funciones POSIX para procesos en Linux

La interfaz de sistema operativo portátil, por sus siglas en inglés *Portable Operating System Interface (POSIX)*, consiste en una colección de funciones especificadas por el estándar IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos, específicamente el código fuente de Unix, Linux y macOS. Además, POSIX establece las reglas para la portabilidad de programas. La biblioteca proporciona un conjunto de funciones que le permiten administrar procesos y otros recursos del sistema. La denominación oficial del estándar es IEEE Std. 1003, e ISO/IEC-9945.

POSIX define los servicios ofrecidos por Unix/Linux. Así, por ejemplo, para Gestión de procesos tales como creación, eliminación, sincronización y temporización. Además, define los servicios ofrecidos por Unix/Linux para Gestión de archivos, tales como creación y borrado de archivos y directorios, manipulación de archivos especiales, protección de la información, Entrada/-Salida y control. La Tabla VIII.1 muestra las funciones POSIX relacionadas con la función fork().

Tabla VIII.1: Funciones POSIX relacionadas con fork()

Función	Descripción
`fork()`	Crea un nuevo proceso hijo duplicando el proceso padre. El proceso hijo hereda las variables, descriptores de archivo, etc. Sin embargo, tiene un ID de proceso diferente.
`vfork()`	Similar a fork(), pero optimizado para casos donde el hijo ejecutará inmediatamente un programa diferente usando exec().
`exec()`	(Familia de funciones: execl(), execv(), execle(), execve(), etc.). Reemplaza el contenido del proceso actual con un nuevo programa ejecutable.
`wait() y waitpid()`	Permiten que un proceso padre espere la terminación de uno o más procesos hijos creados con fork().
`kill()`	Envía una señal a un proceso. Puede ser usado para terminar un proceso hijo creado por fork().
`setsid()`	Crea una nueva sesión dentro del sistema, sirve para que un programa se "independice" del resto. Se usa en su mayoría en demonios después de un fork().
`exit()`	Termina un proceso. El proceso hijo puede finalizarse con esta función.
`getppid()`	Devuelve el ID del proceso padre del proceso actual.

Funciones de identificación de procesos en C para Linux

En los sistemas UNIX/Linux, se disponen de algunas funciones que devuelven los ID del proceso que los llama. Por ejemplo:

- **$ getpid():** devuelve el ID del proceso que llama.
- **$ getuid():** sirve para saber qué usuario es el dueño del programa que se está ejecutando.
- **$ getpgrp():** devuelve el ID del grupo al que pertenece el proceso actual.
- **$ getppid()**: devuelve el ID del proceso padre del proceso que llama. Normalmente es el ID del proceso que creó este proceso utilizando la función fork().

Cabe señalar que *PID (Process ID)* es un identificador único para cada uno de los procesos que se ejecutan en el sistema operativo Linux, el cual se conforma de hasta 5 números enteros. Este identificador desempeña un papel clave en los procesos de Linux, ya que proporciona información que facilita algunas tareas de administración y seguridad del sistema. El identificador de proceso

principal *(PPID)* ayuda a identificar procesos demonios. Cuando el sistema operativo inicia un proceso, hereda un PPID de su proceso principal, generalmente un shell o un proceso del sistema. A continuación, en el siguiente programa en C, se muestran las funciones de identificación de la información de procesos en Linux:

Funciones de información de procesos en C para Linux

Descripción: Este programa despliega en pantalla los ID de la información de los procesos.

```
1  //Program:  Despliegue de los IDs del proceso padre e hijo.
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  int main()
6  {
7     printf("El PID del padre es: %d \n",(int)getppid());
8     printf("El PID del hijo es: %d \n",(int)getpid());
9     printf("El UID del proceso es: %d \n",(int)getuid());
10    printf("El GID del proceso es: %d \n",(int)getgid());
11    return 0;
12 }
13 //I love Linux
```

Listing VIII.3: Programa: Identificadores de procesos

Función fork

La llamada a la función *fork()* se utiliza para crear un nuevo proceso en los sistemas operativos Linux/Unix llamado proceso hijo, que se ejecuta simultáneamente con el proceso que realiza la llamada *fork()* (proceso padre). Luego de crear un nuevo proceso hijo, ambos procesos ejecutarán la siguiente instrucción después de la llamada a la función *fork()*. Cabe destacar que el proceso hijo, utiliza el mismo PC (contador de programa), registros de CPU y archivos abiertos que se utilizaron en el proceso padre. Además, no toma parámetros y devuelve un valor entero.

Se usa para crear un nuevo proceso en Linux llamado proceso hijo el cual se ejecuta de manera concurrente con el proceso padre. Existen diferentes valores que puede retornar la función fork():

- **Valor negativo:** Cuando la creación de un proceso hijo ha fallado.
- **Cero:** Cuando se crea el proceso hijo.
- **Valor positivo:** Tiene el ID del nuevo hijo que se creó.

A continuación se muestra el código de programas en C que ilustran el uso de la función fork() de C para Linux:

Programa 1: Función fork() en C para Linux

Descripción: Este programa invoca a tres llamadas del sistema fork().

```
//Program:  Funci n fork(), primera versi n.
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
//I love Linux
```

Listing VIII.4: Programa: Llamada fork()

Programa 2: Función fork() utilizando un bucle for

Descripción: Este programa invoca a tres llamadas del sistema fork() en un bucle for.

```
//Program:  Funcion fork(), con bucle for.
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int i;
    for (i = 1; i <= 3; i++ )
      fork();
      printf("hello\n");
    return 0;
}
//I love Linux
```

Listing VIII.5: Programa: Llamada fork()

Programa 3: Función fork()

Descripción: Un programa que crea la cadena (árbol) de procesos en la figura adjunta.

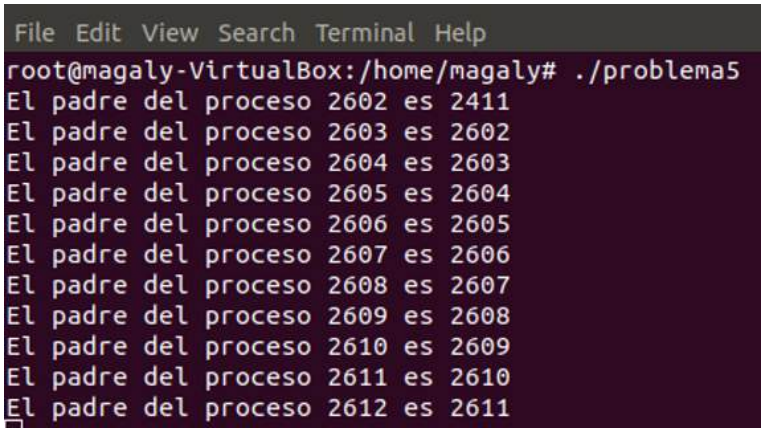
```
//Program:  Funcion fork(), con bucle for.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
  int pid; int n=10;
```

```
9   int i;
10  for (i=0; i<n; i++){
11    pid=fork();
12    if (pid !=0)
13      break;
14  }
15  printf("El padre del proceso %d es %d \n", (int)getpid(), (int)
      getppid());
16  return 0;
17 }
18 // Linux changed my life
```

Listing VIII.6: Programa: Un programa que crea la cadena (árbol) de procesos en la figura adjunta.

Figura VIII.1: Árbol de procesos generado por el programa 3.

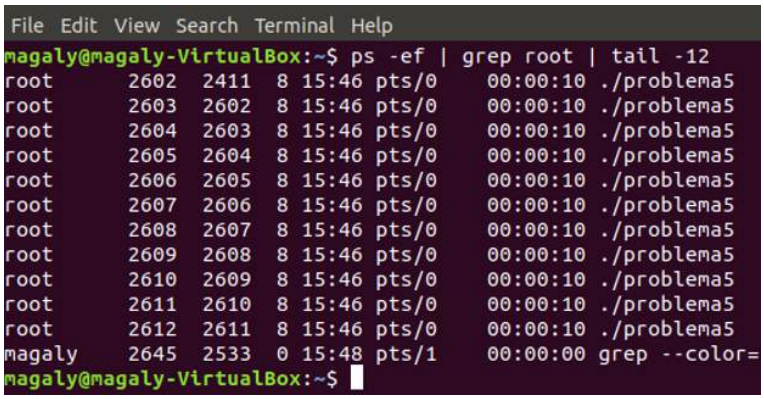
Figura VIII.2: Ejecución del programa 3.



Problemas resueltos

Programa 1

Figura VIII.3: Análisis de los procesos generados por el programa 3.



Descripción: Dado un programa en la llamada al sistema fork(). ¿Cuántos procesos se generarán después de ejecutar el programa anterior?

```c
//Program: La funci n fork()
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork() && fork() || fork();
    fork();
    printf("forked\n");
    return 0;
}
//I love Linux
```

Listing VIII.7: Programa: Función fork de C para Linux

Programa 2

Descripción: Dado un programa en la llamada al sistema fork(). ¿Cuántos procesos se generarán después de ejecutar el programa anterior?

```c
//Program: La funci n fork()
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    pid_t pid;

    if ( (pid=fork()) == 0 )
    { /* hijo */
        printf("Soy el proceso hijo (%d, deciendo de %d)\n", getpid
    (),
        getppid());
    }
    else
    { /* padre */
        printf("Soy el proceso padre (%d, hijo de %d)\n", getpid(),
```

```
        getppid());
    }
     return 0;
}
//I love Linux
```

Listing VIII.8: Programa: Duplicación de procesos mediante fork().

Programa 3

Descripción: Dado un programa que utiliza la función fork(), observar cómo se crea un árbol de procesos con un padre y su hijo. Verifique el subproceso actual durante la ejecución del programa

```
//Program:  rbol  de procesos b sico mediante la llamada a la
   funci n fork()

#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
    pid_t p;
    p = fork();
    if(p<0)
    {
      perror("error de bifurcaci n ");
      exit(1);
    }
    // proceso hijo porque el valor de retorno es cero
    else if ( p == 0)
        printf("Hola desde el Hijo!\n");
   // proceso padre porque el valor de retorno no es cero.
    else
        printf("Hola desde el Padre!\n");
}
int main()
{
    forkexample();
    return 0;
}
```

Listing VIII.9: Programa: Árbol de procesos básico mediante la función fork().

Programa 4

Descripción: Dado un programa que utiliza la función fork(), utilice la llamada waitpid() con la creación de un árbol de procesos con un padre y dos hijos de manera sincronizada.

```
//Program: La funci n fork()
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
```

```
8     pid_t pid1, pid2;
9     int status1, status2;
10
11    if ( (pid1=fork()) == 0 )
12    { /* hijo */
13        printf("Soy el primer hijo (%d, hijo de %d)\n",  getpid(),
    getppid());
14    }
15    else
16    { /*  padre */
17        if ( (pid2=fork()) == 0 )
18        { /* segundo hijo  */
19            printf("Soy el segundo hijo (%d, hijo de %d)\n",  getpid
    (), getppid());
20        }
21        else
22        { /* padre */
23 /* Esperamos al primer hijo */
24            waitpid(pid1, &status1, 0);
25 /* Esperamos al segundo hijo */
26            waitpid(pid2, &status2, 0);
27            printf("Soy el padre (%d, hijo de %d)\n", getpid(),
    getppid());
28        }
29    }
30   return 0;
31 }
32 //I love Linux
```

Listing VIII.10: Programa: Duplicación de procesos mediante fork().

Problemas propuestos

A continuación, se presentan algunos problemas de programación, los cuales se proponen a ser resueltos en lenguaje C, se deberá de utilizar la función fork():

Problema 1: Crear un árbol binario de procesos. Elabore un árbol binario de procesos de tres niveles. El proceso raíz debe ser el proceso principal (padre). Cada proceso (excepto las hojas) debe crear exactamente dos hijos. Los procesos hoja deben imprimir su propio PID y el PID de su padre. Asegúrese de que los procesos hijos terminen antes de que su padre lo haga.

Problema 2: Representación de una jerarquía de tareas. Elabore un programa en C para Linux que llame a la función fork() que simule una jerarquía de tareas con procesos. El proceso principal debe crear dos subprocesos hijos. Cada hijo debe crear un nieto. Los nietos deben crear un bisnieto cada uno. Cada proceso debe imprimir su nivel en el árbol (padre, hijo, nieto, bisnieto), su PID, y el PID de su padre.

Problema 3: Árbol de procesos n-ario. Elabore un programa en C para Linux que llame a la función fork() que construya un árbol de procesos con una estructura N-aria. El número de hijos NN debe ser un argumento del programa pasado por la línea de comandos. Cada proceso debe crear N hijos hasta alcanzar una profundidad de 3 niveles. Cada proceso debe imprimir su PID, el PPID de su padre, y el número de hijos que creó.

Problema 4: Sincronización de un Árbol de Procesos. Desarrolle un programa que genere un proceso padre, un hijo y un nieto. El hijo y el nieto deben realizar una tarea simulada. El padre no debe finalizar hasta que todos los procesos creados hayan terminado, como ayuda puede usar la función wait().

Problema 5: Árbol jerárquico de procesos. Escriba un programa en C que genere un árbol de procesos usando fork(). El proceso principal crea dos hijos y cada uno de estos crea dos hijos más. Cada proceso debe indicar cuándo inicia y cuándo termina su ejecución, todo proceso debe esperar la finalización de sus hijos antes de finalizar.

Conclusiones del capítulo y lecciones aprendidas

Este capítulo ha sido enfocado en aprender los fundamentos de la programación de procesos en C para Linux. Los lectores han aprendido los conceptos básicos y la importancia de la creación de procesos en Linux. Han entendido cómo editar un programa en C, cómo compilarlo y ejecutarlo en una interfaz de línea de comandos en Linux. Se ha explicado los elementos de un proceso en Linux cuando están en ejecución en la RAM. Se ha ilustrado los tipos de procesos en Linux. Se ha puesto énfasis las funciones o llamadas POSIX para el empleo de C para Linux. Además, se han listado sus principales funciones a la hora de utilizarlas durante la programación de procesos. Luego se vincularon estos conocimientos a la gestión de procesos en Linux revisada en el apartado anterior. Se evidenció cómo programar procesos en Linux, especialmente utilizando la función fork() con ejemplos ilustrativos. El aprendizaje se fortaleció mediante los ejercidos resueltos, propuestos y las lecciones aprendidas. Con ello los lectores podrán dar un paso adelante para aprender la programación paralela o concurrente de Hilos en C para Linux.

Lecciones aprendidas

- Un proceso es un programa en ejecución. Un proceso es una entidad que todo sistema operativo utiliza para controlar el uso de los recursos computacionales.
- Un proceso en memoria principal cuando está en estado de ejecución se constituye del código en texto, la data, el heap; y la pila.
- En sistemas operativos, es necesario considerar la jerarquía de los procesos que se están ejecutando. Un proceso iniciado por una aplicación de software o un comando es un proceso padre.
- La programación de procesos en Linux implica la creación y gestión de procesos, que son instancias en ejecución de programas.
- GCC es un conjunto de compiladores desarrollado por el Proyecto GNU.
- POSIX consiste en una colección de funciones especificadas por el estándar IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos, específicamente el código fuente de Unix, Linux y macOS.
- La llamada a la función *fork()* se utiliza para crear un nuevo proceso en los sistemas operativos Linux/Unix llamado proceso hijo, que se ejecuta simultáneamente con el proceso que realiza la llamada *fork()* (proceso padre)

Sitios de interés

(I) Creación y duplicación de procesos en C (Parte I) – Linux: Propuesto por Rodrigo Paszniuk | 2013-08-17: Podcast

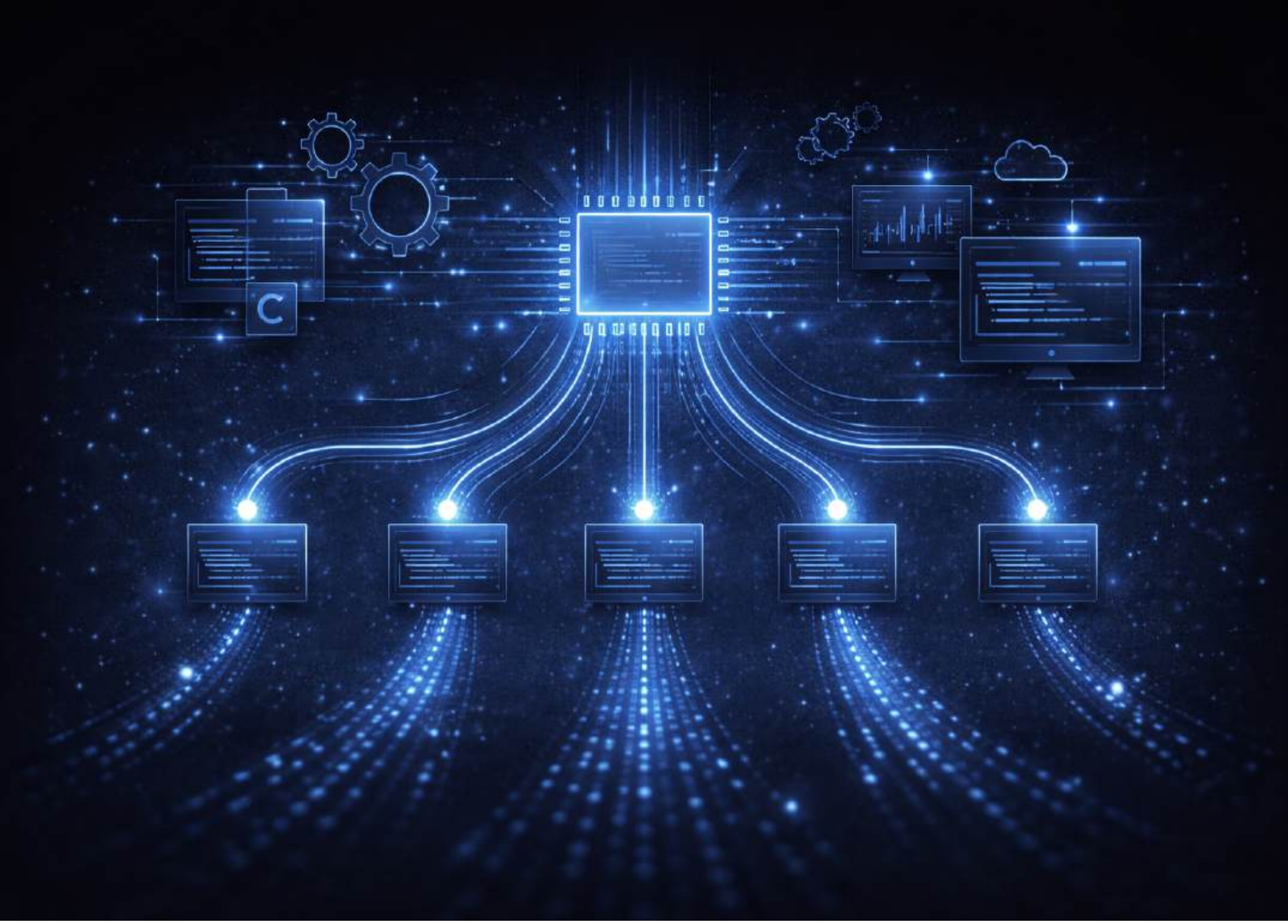
(II) S. Operativos | | Árboles de Procesos con Función fork()(Ejemplo 1): Propuesto por Gus IM Video de YouTube

(III) fork(). Parte I: Creación de un nuevo proceso, hijos, padres, zombies y huérfanos. Propuesto por: WhileTrueThenDoctor. Video en YouTube

Autoevaluación del Capítulo

1. **¿Qué hace la función `fork()` en un programa en C?**
   - Crea un hilo y el tiempo de vida que le queda.
   - Crea un nuevo proceso duplicando el proceso padre.
   - Termina el proceso actual y libera la memoria almacenada.
   - Detiene temporalmente el proceso.
2. **¿Qué valor o respuesta devuelve la función `fork()` en el proceso hijo?**
   - -1
   - 0
   - El PID del proceso hijo.
   - Ninguno de los anteriores.
3. **¿Qué valor devuelve la función `fork()` en el proceso padre?**
   - -1
   - 0
   - El PID del proceso hijo.
   - Ninguno de los anteriores.
4. **¿Qué sucede si la función `fork()` falla en medio de la ejecución?**
   - El programa se ejecuta infinitamente.
   - Devuelve -1.
   - Se reinicia.
   - Termina todos los procesos y devuelve 0.
5. **¿Qué función POSIX se utiliza para esperar a que un proceso hijo termine?**
   - `vfork()`
   - `exec()`
   - `wait()`
   - `kill()`

6. **¿Qué función permite reemplazar el código de un proceso después de un `fork()`?**
   - `waitpid()`
   - `getpid()`
   - `kill()`
   - `exec()`
7. **¿Qué función devuelve el ID del proceso actual?**
   - La devuelve `getppid()`
   - Lo trae `wait()`
   - La devuelve `getpid()`
   - Lo trae `exit()`
8. **¿Para qué sirve la función `vfork()` dentro de Linux?**
   - Genera un hilo sin más.
   - Finaliza un proceso de forma segura.
   - Optimiza la creación de procesos cuando el hijo ejecutará un `exec()`.
   - Sincroniza procesos en memoria.
9. **¿Cuál es la diferencia entre `fork()` y `vfork()`?**
   - `vfork()` el hijo utiliza temporalmente los recursos del padre hasta entrar en `exec()`.
   - `fork()` puede que falle cuando no hay memoria suficiente.
   - `vfork()` crea un hilo en lugar de un proceso.
   - No hay diferencias significativas.
10. **¿Qué función permite enviar señales a un proceso hijo creado con `fork()` para eliminarlo?**
    - `exec()`
    - `wait()`
    - `kill()`
    - `exit()`



CAPÍTULO IX

Hilos

Introducción del Capítulo

Este capítulo mostrará el uso de hilos en lenguaje C dentro de un entorno Linux. Los hilos son componentes que permiten la ejecución simultánea de tareas, optimizando el rendimiento de aplicaciones en sistemas multitarea y de alto rendimiento. Se explorarán conceptos clave como la creación, gestión y sincronización de hilos, proporcionando una base teórica sólida y ejemplos prácticos.

Los temas cubiertos muestran la idea y clases de hilos, las distinciones entre programación concurrente y paralela, y también los modelos de procesos basados en un solo hilo y en varios hilos. Aparte del uso de la Biblioteca de Hilos POSIX, norma muy usada en la programación de hilos en C, junto con los tipos de datos y funciones importantes, los modos de creación y las formas de sincronización.

El capítulo presenta asuntos resueltos que muestran el uso real de las ideas, al igual que tareas sugeridas hechas para mejorar las destrezas obtenidas. Esta visión completa ayudará a entender no solo las bases clave, sino también los retos técnicos unidos al manejo de bienes comunes y a la unión entre hilos.

Tras acabar este capítulo, el lector habrá logrado comprender el uso de los hilos y estará listo para poner en marcha respuestas eficaces en el mundo de sistemas operativos actuales y programas de software. Con este saber, podrá usar al máximo los bienes computacionales y hacer programas que saquen provecho de las estructuras multinúcleo de hoy.

¡Controla tus procesos, controla tu sistema!
¡Bienvenidos!

Fundamentos

El uso de hilos permite a las aplicaciones realizar múltiples tareas de manera simultánea, mejorando el rendimiento y la experiencia del usuario. La programación con hilos se centra en la capacidad de dividir una tarea grande en subtareas más pequeñas que pueden ejecutarse en paralelo.

Concepto de hilos de software

Como se ha señalado, los hilos fueron diseñados para lograr la programación concurrente o paralela. Esto es posible gracias a la implementación de un **Hilo por software**. Un hilo es una propiedad que permite a los sistemas operativos ejecutar una aplicación de software realizando varias tareas independientes a la vez. En la comunidad académica se habla la siguiente taxonomía de hilos: un proceso un hilo, varios procesos, varios hilos, un proceso varios hilos y varios procesos, varios hilos.

Tipos de hilos

Los hilos pueden clasificarse según su origen y características principales, por lo cual es importante aprender a reconocerlos, para de esa forma saber cuál usar:

- **Hilos de usuario:** son hilos que el programa crea y controla por su cuenta sin que el sistema operativo se encargue directamente de ellos.
- **Hilos de núcleo:** El sistema operativo gestiona el espacio de los hilos que se ejecutan en núcleo, y él los gestiona.

Tipos de Programación de Hilos

La programación de hilos se puede clasificar en la **programación concurrente** y la **programación paralela**, dependiendo del objetivo y la arquitectura que se maneje.

Programación Concurrente

Consiste en organizar un programa para que pueda manejar varias tareas al mismo tiempo. En realidad, en un solo procesador las tareas se van turnando rápidamente, dando la impresión de que ocurren simultáneamente.

- Ejemplo: Una aplicación mediante un browser que proporciona varias API de audio, video, y procesamiento de datos a la vez.
- Ventaja: Mayor reactividad y utilización eficiente del tiempo ocioso.
- Reto: gestionar discretamente las necesidades de sincronización y el acceso simultáneo a recursos computacionales.

Programación Paralela

Consiste en ejecutar varias tareas al mismo tiempo de verdad, usando varios núcleos o procesadores. Esto permite que el programa termine más rápido porque varias tareas avanzan simultáneamente.

- Ejemplo: Un programa usa varios hilos; uno calcula datos mientras otro escribe resultados en un archivo.
- Ventaja: Permite usar varios núcleos del procesador y terminar antes.
- Desafío: Controlar que varios hilos no usen el mismo recurso al mismo tiempo.

Modelo de Procesos de Hilos

Modelo de Procesos de un Único Hilo

En este modelo, cada proceso tiene un solo hilo de ejecución. Todas las tareas del proceso se ejecutan secuencialmente en un único flujo de control. Este modelo simplifica el diseño, pero limita el rendimiento en sistemas multicore.

Modelo de Procesos Multihilo

Cuando un proceso pesado que está en ejecución está compuesto por varios hilos que comparten un determinado espacio de memoria y que pueden correr diferentes tareas independientes cada uno, como consecuencia, se optimiza la eficiencia y el rendimiento del sistema computacional.

Programación de Hilos en C

Biblioteca de Hilos POSIX

La biblioteca POSIX Threads (`pthread`) es un estándar para la programación de hilos en sistemas operativos tipo UNIX. Incluye funciones para la creación, gestión y sincronización de hilos.

En el Lenguaje C ANSI, dentro del estándar POSIX, se debe incluir la librería de cabecera **<pthread.h>** para disponer de las diferentes funciones de gestión de los hilos. Para compilarlo, y verificar que no existan errores se debe añadir la opción **-pthread**.

Tipos de Datos en Hilos

Los hilos en el estándard POSIX emplean una estructura específica que permite programar y gestionar los hilos **pthread_t**. En la Tabla IX.1 se listan los tipos de datos de hilos y la descripción de la actividad que cumplen.

Tabla IX.1: Tipos de Datos pthread

Tipo de Datos	Descripción
`pthread_attr_t`	Define los atributos de configuración de un hilo.
`pthread_mutexattr_t`	Permite configurar los atributos de un mutex (bloqueo de exclusión mutua).
`pthread_condattr_t`	Proporciona atributos para variables de condición.
`pthread_mutex_t`	Representa un mutex utilizado para sincronizar el acceso a recursos compartidos.
`pthread_cond_t`	Una variable de condición utilizada para la sincronización entre hilos.
`pthread_t`	Identificador único para cada hilo.
`pthread_once_t`	Asegura que una acción se ejecute únicamente una vez en un programa multihilo.
`pthread_key_t`	Clave utilizada para acceder a datos específicos de un hilo.

Tipos de Funciones en Hilos

La biblioteca `pthread` proporciona varias funciones clave:

Tabla IX.2: Funciones clave en la biblioteca POSIX Threads

Función	Descripción
`pthread_create`	Crea un nuevo hilo.
`pthread_exit`	Termina la ejecución de un hilo.
`pthread_join`	Espera la terminación de un hilo.
`pthread_mutex_init`	Inicializa un mutex para sincronización.
`pthread_mutex_lock`	Bloquea un mutex (exclusión mutua).
`pthread_mutex_unlock`	Desbloquea un mutex.
`pthread_cond_wait`	Espera una señal en una variable de condición (usada con un mutex).
`pthread_cond_signal`	Envía una señal a un hilo esperando en una variable de condición.

Creación de hilos en C

La creación de hilos es una de las funciones principales de la biblioteca POSIX Threads. Un hilo permite dividir el programa en tareas que puedan ejecutarse de manera simultánea.

Pasos para crear un hilo:

1. Incluir la biblioteca pthread
2. Definición de la función del hilo
3. Declaración de un identificador de un hilo
4. Creación del hilo

Ejemplo básico de creación de hilos:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

/* Definicion de la funcion del hilo */
void *threadFunction(void *args) {
    printf("I am thread-Function.\n");
    return NULL;
}

int main() {
    /* Declaracion del identificador del hilo */
    pthread_t id;
    int ret;

    /* Creacion del hilo */
```

```
    ret = pthread_create(&id, NULL, &threadFunction, NULL);
    if (ret == 0) {
        printf("Thread created successfully.\n");
        usleep(1000000); // Retraso de 1 segundo
    } else {
        printf("Thread not created.\n");
        return 0; /* Salir si la creaci n falla */
    }

    printf("I am main function.\n");
    return 0;
}
```

Listing IX.1: Creación de un hilo en C

Sincronización de Hilos

La sincronización de hilos es crucial cuando varios hilos acceden a recursos compartidos. Sin mecanismos de sincronización, pueden surgir conflictos, como condiciones de carrera, donde los resultados dependen del orden de ejecución de los hilos. La biblioteca POSIX Threads proporciona herramientas para manejar estos problemas, como los **mutex**.

¿Definición de exclusividad mutua (mutex)

Cuando se requiere que un determinado recurso compartido, sea un dato, un registro, un contador, una señal, un valor o una cifra, deba ser utilizado por cualquier hilo en ejecución, accediendo a una zona crítica, es indispensable que un mecanismo de sincronización evite conflictos o problemas. De esa manera el sistema operativo puede certificar que un solo hilo puede acceder a dicho recurso compartido a la vez.

El procedimiento podría ser el siguiente:

1. Incorporar la biblioteca de gestión de hilos **pthread**.
2. Inicializar la exclusión mutua.
3. Bloquear la zona crítica antes de que sea accedida a un recurso compartido.
4. Acceder a la zona crítica y editar el recurso compartido.
5. Desbloquear la zona y actualizar el recurso.
6. Destruir la zona crítica (mutex) al finalizar la ejecución del hilo programa.

Ejemplo básico de sincronización con mutex:

```c
#include <stdio.h>
#include <pthread.h>

/* Declaracion del mutex */
pthread_mutex_t mutex;
int contador = 0;

/* Funcion ejecutada por los hilos */
void *incrementar(void *arg) {
    pthread_mutex_lock(&mutex); // Bloquear el mutex
    contador++;
    printf("Contador: %d\n", contador);
    pthread_mutex_unlock(&mutex); // Desbloquear el mutex
    return NULL;
}

int main() {
    pthread_t hilo1, hilo2;

    /* Inicializar el mutex */
    pthread_mutex_init(&mutex, NULL);

    /* Crear hilos */
    pthread_create(&hilo1, NULL, incrementar, NULL);
    pthread_create(&hilo2, NULL, incrementar, NULL);

    /* Esperar a que los hilos terminen */
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    /* Destruir el mutex */
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Listing IX.2: Sincronización de hilos con mutex

Problemas resueltos

A continuación, se presentan ejemplos prácticos que muestran cómo resolver problemas comunes utilizando hilos y sincronización:

1. **Ejemplo 1: Imprimir mensajes simultáneamente**
   Crear dos hilos que impriman los mensajes `Hola` y `Mundo` intercaladamente:

```c
#include <stdio.h>
#include <pthread.h>

void *imprimirHola(void *arg) {
    printf("Hola ");
    return NULL;
```

```
}

void *imprimirMundo(void *arg) {
    printf("Mundo\n");
    return NULL;
}

int main() {
    pthread_t hilo1, hilo2;
    pthread_create(&hilo1, NULL, imprimirHola, NULL);
    pthread_create(&hilo2, NULL, imprimirMundo, NULL);

    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);
    return 0;
}

```

Listing IX.3: Imprimir mensajes con hilos

2. **Ejemplo 2: Incrementar un contador compartido con un mutex**
   Este ejemplo demuestra cómo usar un **mutex** para evitar problemas cuando varios hilos acceden a la misma variable compartida. Sin un mutex, dos hilos podrían intentar incrementar el contador al mismo tiempo, causando resultados incorrectos.

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;
int contador = 0;

void *incrementar(void *arg) {
    pthread_mutex_lock(&mutex);
    contador++;
    printf("Contador: %d\n", contador);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main() {
    pthread_t hilo1, hilo2;
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&hilo1, NULL, incrementar, NULL);
    pthread_create(&hilo2, NULL, incrementar, NULL);
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    pthread_mutex_destroy(&mutex);
    return 0;
}

```

Listing IX.4: Contador sincronizado con mutex

Conclusiones del capítulo

Este capítulo ha proporcionado una introducción práctica y conceptual al uso de hilos en la programación, específicamente en el lenguaje C y dentro del contexto de los sistemas operativos modernos. Los lectores han aprendido cómo los hilos permiten dividir tareas en subtareas más pequeñas y eficientes, optimizando el uso de los recursos computacionales. Se han explorado conceptos fundamentales como la creación, gestión y sincronización de hilos mediante la biblioteca POSIX Threads, que proporciona herramientas esenciales para la programación multihilo. Además, se han presentado las diferencias entre la programación concurrente y paralela, así como los modelos de procesos de un solo hilo y multihilo, destacando sus ventajas y desafíos. Los ejemplos prácticos y problemas resueltos proporcionados a lo largo del capítulo han permitido consolidar estos conocimientos, ayudando al lector a desarrollar habilidades para diseñar programas eficientes y seguros que aprovechen la arquitectura multinúcleo de los sistemas modernos. Al finalizar este capítulo, el lector no solo comprende los fundamentos de los hilos, sino que también está preparado para aplicar estos conceptos en proyectos prácticos, abordando desafíos técnicos como la sincronización y la concurrencia de manera efectiva.

Lecciones aprendidas

- La biblioteca POSIX Threads incluye funciones para la creación, gestión y sincronización de hilos.
- En el Lenguaje C ANSI, dentro del estándar POSIX, se debe incluir la librería de cabecera pthread.h para disponer de las diferentes funciones de gestión de los hilos. Para compilarlo, y verificar que no existan errores se debe añadir la opción -pthread.
- Los hilos en el estándar POSIX emplean una estructura específica que permite programar y gestionar los hilos pthread_t.
- Existen grandes diferencias entre la programación concurrente y la programación paralela, ya que la primera simula simultaneidad en un solo núcleo, la segunda distribuye tareas reales en varios núcleos.
- Los modelos de procesos de un solo hilo y multihilo presentan ventajas y desventajas dependiendo de las necesidades de la aplicación.

Sitios de interés

Para ampliar los conocimientos adquiridos en este capítulo, se recomiendan los siguientes recursos en línea y bibliográficos:

- **Documentación oficial de pthreads:** `https://man7.org/linux/man-pages/man7/pthreads.7.html`. Una guía detallada y oficial sobre la biblioteca POSIX Threads.
- **Tutorial de programación multihilo en C - Guías para multithreading:** `https://www.geeksforgeeks.org/multithreading-in-c/`. Este recurso ofrece ejemplos prácticos, explicaciones y un paso a paso para el manejo de hilos.
- **Libro: Sistema operativo Linux**, por Ricardo Castillo, noviembre de 2020. Repositorio de la universidad de San Marcos, Lima Perú, que caracteriza a los sistemas operativos basados en Linux como aquellos que proveen mayor seguridad de la información que sus competencias.

- **Artículos de IBM Developer:** `https://developer.ibm.com`. Contiene publicaciones avanzadas y tutoriales sobre sistemas operativos, incluyendo hilos y concurrencia.
- **Stack Overflow:** `https://stackoverflow.com`. Una comunidad activa donde puedes encontrar respuestas a dudas específicas sobre la implementación de hilos en C.

Autoevaluación del capítulo

Responde las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **Seleccione la ventaja de usar hilos en una aplicación dentro de Linux**
   - Reducir el uso de memoria al codificar.
   - Facilitan la ejecución concurrente o paralela de una aplicación de software
   - Forman parte del hardware de los procesos de los sistemas computacionales.
   - Incrementan el overhead del CPU.
2. **¿Qué biblioteca se utiliza para la programación de hilos en C en sistemas UNIX/Linux?**
   - `stdthread`.
   - `pthread`.
   - `threading.h`.
   - Ninguna de las anteriores.
3. **¿Qué función de pthreads se utiliza para esperar la terminación de un hilo?**
   - `pthread_wait`.
   - `pthread_exit`.
   - `pthread_join`.
   - `pthread_kill`.

4. **¿Cuál de las siguientes afirmaciones es correcta sobre la sincronización con mutexs?**

   - Los mutexs garantizan que varios hilos puedan ejecutar una sección crítica simultáneamente.
   - Un mutex asegura que un solo hilo puede acceder a un recurso compartido a la vez.
   - Los mutexs son análogos a los semáforos en sistemas operativos.
   - Los mutexs son tecnología de programación actual.

5. **¿Qué ocurre si no se usa un mecanismo de sincronización para proteger el acceso a un recurso compartido entre hilos?**

   - El productor y el consumidor no tendrían límites de ejecución.
   - A pesar de esa señal se terminaría con dicha ejecución.
   - El sistema operativo no encontraría la manera técnica de cuidar la zona crítica.
   - Los procesos de multi-tarea buscarían otro mecanismo para su ejecución.

6. **¿Qué pasa con un hilo si se llama a `pthread_exit`?**

   - Continúa corriendo.
   - Espera ser llamado por otro hilo.
   - El hilo termina su ejecución.
   - No ocurre nada.

7. **¿Qué función se utiliza para inicializar un mutex antes de usarlo?**

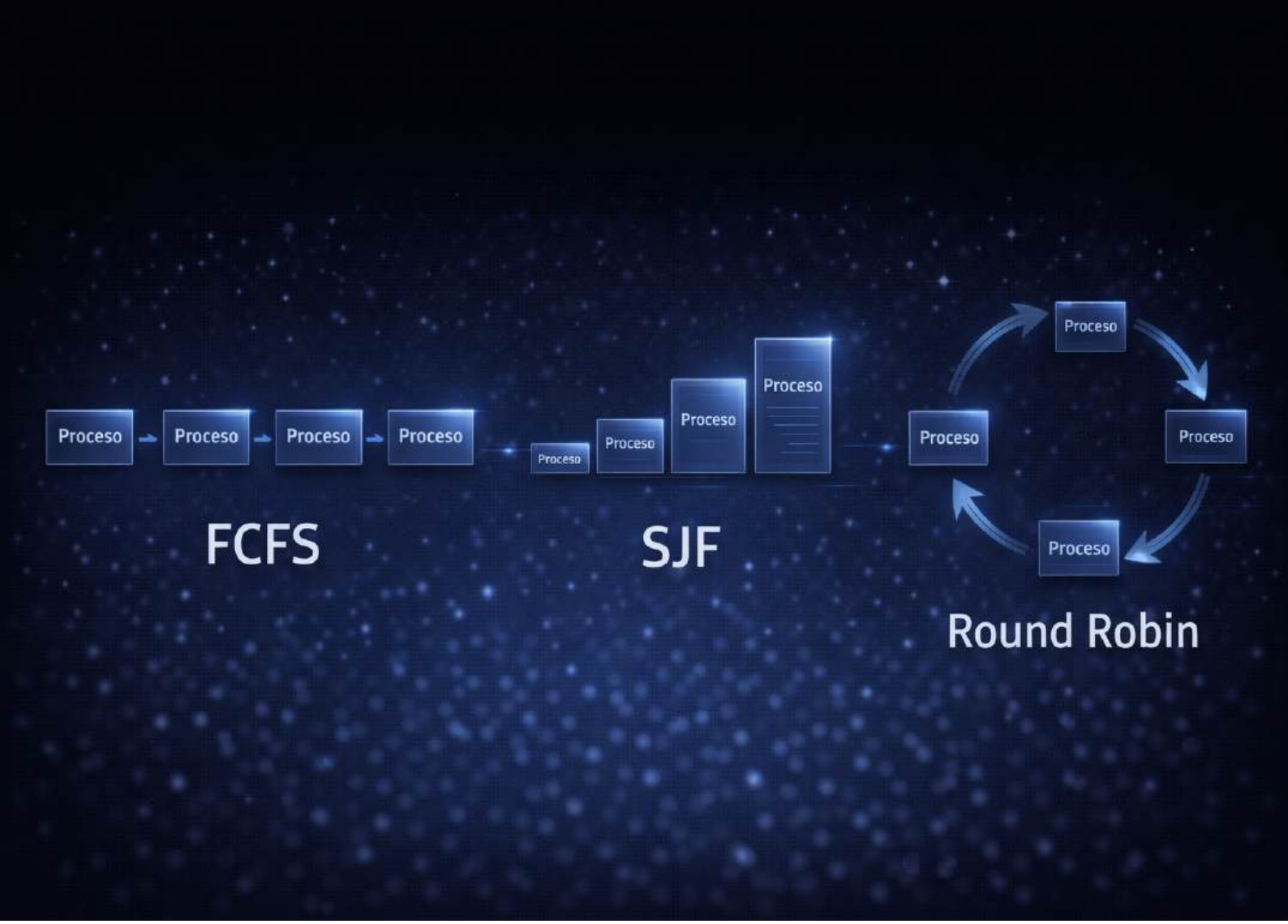
   - `pthread_start`.
   - `pthread_destroy`.
   - `pthread_mutex_init`.
   - `pthread_setup`.

8. **¿Cuál es el principal problema que resuelve el uso de variables de condición (`pthread_cond_t`)?**

   - La comunicación entre procesos, la cual es muy compleja de llevar.

- La sincronización entre hilos basada en condiciones.
- Inicia el hilo con datos almacenados.
- Prioriza tareas.

9. **En un programa multihilo, ¿cuál de las siguientes situaciones puede generar un bloqueo (deadlock)?**
   - `pthread_exit` llamado antes de `pthread_create`.
   - Dos hilos comparten datos sin sincronización.
   - Dos hilos intentan bloquearse mutuamente con dos mutexes diferentes.
   - No se utiliza `pthread_join`.
10. **¿Qué ventaja tienen los hilos sobre el uso de múltiples procesos?**
    - Los hilos no consumen recursos
    - Los hilos son más seguros
    - Los hilos comparten el mismo espacio de memoria
    - Los hilos no se sincronizan dos veces.



CAPÍTULO X

Algoritmos de Planificación de Procesos

Introducción

La planificación de procesos es un componente esencial en los sistemas operativos, ya que determina el orden en que los procesos se ejecutan en el procesador. El objetivo principal de los algoritmos de planificación es maximizar la eficiencia del sistema, equilibrando el uso del procesador y minimizando métricas como el tiempo de espera y el tiempo de retorno [117].

Los sistemas operativos tienen la función de gestionar la cola de procesos. Para todos es conocido que la microelectrónica junto con la informática ha diseñado procesadores para ejecutar procesos concurrentemente. Para optimizar esta función, la industria y la comunidad científica han desarrollado varios algoritmos de planificación de procesos, los cuales deben establecer el orden o prioridad de ejecución [27].

Este capítulo aborda los principales algoritmos, métricas e indicadores para la planificación de procesos, explica su funcionamiento y sus efectos en el rendimiento del sistema. Se destacan métricas como el tiempo de espera, el tiempo de respuesta y el tiempo de retorno, lo que permite medir el impacto de cada estrategia de planificación. Se establecieron máquinas virtuales con Linux como plataforma experimental.

Se revisarán algoritmos como First-Come, First-Served (FCFS), Shorest Job First (SJF) y "Round Robin". Los ejemplos y ejercicios presentes permitirán comprender sus implicaciones para la gestión de procesos. Mediante la implementación en C, identificado sus ventajas y limitaciones.

"Date tiempo para planificar, porque así tendrás tiempo para todo lo demás."
Miguel Ángel Cornejo.
¡Bienvenidos!

Métricas e Indicadores

Las métricas de rendimiento son fundamentales para evaluar la efectividad de un algoritmo de planificación. Entre las principales métricas se encuentran:

- **Tiempo de espera**: Tiempo que un proceso pasa en la cola de espera antes de ser ejecutado [124].
- **Tiempo de retorno**: Tiempo total desde que un proceso entra en el sistema hasta que finaliza [127].
- **Tiempo de respuesta**: Tiempo transcurrido desde que un proceso realiza una solicitud hasta que recibe la primera respuesta [31].
- **Uso de la CPU**: Porcentaje del tiempo en el que el procesador está ocupado [52].
- **Rendimiento**: Es la medida del tiempo de uso de la CPU versus los recursos empleados durante la ejecución de un proceso [16].

Tipos de Algoritmos

En el siguiente apartado se explican algunos tipos de algoritmos de planificación.

FCFS (First-Come, First-Served)

El algoritmo FCFS asigna la CPU a los procesos en el orden en que llegan a la cola de espera. Este enfoque es sencillo pero puede llevar a tiempos de espera elevados en sistemas con procesos de duración variable [117] (Véase Tabla X.1.

Tabla X.1: Ejemplo del algoritmo FCFS [124]

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Paso a paso:

1. Los procesos llegan en los tiempos 0, 1, 2 y 3 respectivamente.

2. Se ejecutan en el orden de llegada: P1, P2, P3 y P4.

3. Tiempo de espera:

   - P1: 0 (se ejecuta inmediatamente).
   - P2: 5 (espera mientras P1 se ejecuta).
   - P3: 8 (espera mientras P1 y P2 se ejecutan).
   - P4: 16 (espera mientras P1, P2 y P3 se ejecutan).

4. Tiempo de retorno:

   - P1: 5, P2: 8, P3: 16, P4: 22.

Programación de FCFS

En el siguiente apartado, se muestra el código en lenguaje C para Linux dé dos algoritmos de planificación de procesos en el orden "el primero que entre, el primero en ser atendidos (FCFS)". El primer algoritmo, diseñado especialmente para esta obra académica. El segundo, recopilado de un repositorio público de GitHub [63], que será utilizado para realizar evaluaciones comparativas en su estructura, actuación y resultados. .

- **Código FCFS - Libro**

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct process {
    int pn;
    float at, bt, ct, tat, wt;
    struct process *next;
} process;

float total_wait_time = 0, total_turn_around_time = 0;
int n;

process *get_process(float a, float b) {
    static int i = 1;
    process *pr = (process *)malloc(sizeof(process));
    pr->at = a, pr->bt = b;
    pr->pn = i++;
    pr->next = NULL;
    return pr;
}

void insert(process *ptr, float a, float b) {
    process *pro, *temp;
    pro = get_process(a, b);
```

```
25      while (ptr->next != NULL) {
26          temp = ptr;
27          ptr = ptr->next;
28          if (ptr->at > a) {
29              pro->next = ptr;
30              temp->next = pro;
31              return;
32          }
33      }
34      ptr->next = pro;
35  }
36
37  void display(process *ptr) {
38      process *temp = ptr->next;
39      printf("Process\tArrival time\tBurst time\tCompletion time\
        tWaiting time\tTurn around time\n");
40      while (temp != NULL) {
41          printf("P[%d]\t%3.2f\t\t%3.2f\t\t%3.2f\t\t%3.2f\t\t%3.2
        f\n", temp->pn, temp->at, temp->bt, temp->ct, temp->wt, temp
        ->tat);
42          temp = temp->next;
43      }
44      printf("Average waiting time: %3.2f\nAverage turn around
        time: %3.2f\n", total_wait_time / n, total_turn_around_time
        / n);
45  }
46
47  void ganttchart(process *ptr) {
48      process *temp, *t;
49      temp = ptr->next;
50      t = ptr;
51      while (temp != NULL) {
52          temp->ct = t->ct + temp->bt;
53          t = temp;
54          temp = temp->next;
55      }
56  }
57
58  void wt_and_tat(process *ptr) {
59      process *temp = ptr->next;
60      while (temp != NULL) {
61          temp->tat = temp->ct - temp->at;
62          total_turn_around_time += temp->tat;
63          temp->wt = temp->tat - temp->bt;
64          total_wait_time += temp->wt;
65          temp = temp->next;
66      }
67  }
68
69  int main() {
70      int i;
71      float a, b;
72      process *START = (process *)malloc(sizeof(process));
73      START->next = NULL;
74      printf("=========FCFS Algorithm==========\nEnter the number
        of processes: ");
```

```
75      scanf("%d", &n);
76      for (i = 0; i < n; i++) {
77          printf("Process P(%d)\n", i + 1);
78          printf("Arrival time? ");
79          scanf("%f", &a);
80          printf("Execution time? ");
81          scanf("%f", &b);
82          insert(START, a, b);
83          printf("\n");
84      }
85      ganttchart(START);
86      wt_and_tat(START);
87      display(START);
88      return 0;
89  }
90
```

Listing X.1: Código FCFS 1 (Libro)

- **Implementación referenciada desde GitHub**

  El repositorio público [99] incluye una versión funcional del algoritmo FCFS en lenguaje C.

```
1  #include<stdio.h>
2  struct proc {
3      int no, at, bt, ct, tat, wt;
4  };
5
6  struct proc read(int i) {
7      struct proc p;
8      printf("\nProcess No: %d\n", i);
9      p.no = i;
10     printf("Enter Arrival Time: ");
11     scanf("%d", &p.at);
12     printf("Enter Burst Time: ");
13     scanf("%d", &p.bt);
14     return p;
15 }
16
17 int main() {
18     struct proc p[10], tmp;
19     float avgtat = 0, avgwt = 0;
20     int n, ct = 0;
21     printf("<--FCFS Scheduling Algorithm (Non-Preemptive)-->\n"
       );
22     printf("Enter Number of Processes: ");
23     scanf("%d", &n);
24     for (int i = 0; i < n; i++) {
25         p[i] = read(i + 1);
26     }
27     for (int i = 0; i < n - 1; i++) {
28         for (int j = 0; j < n - i - 1; j++) {
29             if (p[j].at > p[j + 1].at) {
30                 tmp = p[j];
31                 p[j] = p[j + 1];
```

```
32                  p[j + 1] = tmp;
33              }
34          }
35      }
36      printf("\nProcessNo\tAT\tBT\tCT\tTAT\tWT\tRT\n");
37      for (int i = 0; i < n; i++) {
38          ct += p[i].bt;
39          p[i].ct = ct;
40          p[i].tat = p[i].ct - p[i].at;
41          avgtat += p[i].tat;
42          p[i].wt = p[i].tat - p[i].bt;
43          avgwt += p[i].wt;
44          printf("P%d\t\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].no, p[i
    ].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt, p[i].wt);
45      }
46      avgtat /= n;
47      avgwt /= n;
48      printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f
    ", avgtat, avgwt);
49      return 0;
50  }
51
```

Listing X.2: Código FCFS 2 (Github)

SJF (Shortest Job First)

El algoritmo SJF selecciona el proceso con el menor tiempo de ráfaga disponible. Este enfoque es óptimo en términos de tiempo de espera promedio, sin embargo, depende de conocer los tiempos de ráfaga con precisión [127]. (Véase Tabla X.2).

Tabla X.2: Ejemplo del algoritmo SJF [31]

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	6
P2	1	4
P3	2	5
P4	3	3

Paso a paso:

1. Los procesos llegan en los tiempos 0, 1, 2 y 3.
2. Siempre se selecciona el proceso con menor duración disponible.
3. Orden de ejecución:
   - P4, P2, P3 y finalmente P1.
4. Tiempos de espera calculados:

- P4: 0, P2: 3, P3: 7, P1: 12.

5. Tiempos de retorno calculados:

   - P4: 3, P2: 7, P3: 12, P1: 18.

Implementación del Algoritmo SJF

El siguiente código implementa el algoritmo SJF no expropiativo:

```
#include<stdio.h>
struct proc {
    int no, at, bt, it, ct, tat, wt;
};

struct proc read(int i) {
    struct proc p;
    printf("\nProcess No: %d\n", i);
    p.no = i;
    printf("Enter Arrival Time: ");
    scanf("%d", &p.at);
    printf("Enter Burst Time: ");
    scanf("%d", &p.bt);
    return p;
}

int main() {
    int n, j, min = 0;
    float avgtat = 0, avgwt = 0;
    struct proc p[10], temp;
    printf("<--SJF Scheduling Algorithm (Non-Preemptive)-->\n");
    printf("Enter Number of Processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        p[i] = read(i + 1);
    }
    for (int i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
    for (j = 1; j < n && p[j].at == p[0].at; j++) {
        if (p[j].bt < p[min].bt) {
            min = j;
        }
    }
    temp = p[0];
    p[0] = p[min];
    p[min] = temp;
    p[0].it = p[0].at;
    p[0].ct = p[0].it + p[0].bt;
```

```
46      for (int i = 1; i < n; i++) {
47          for (j = i + 1, min = i; j < n && p[j].at <= p[i - 1].ct; j
    ++) {
48              if (p[j].bt < p[min].bt) {
49                  min = j;
50              }
51          }
52          temp = p[i];
53          p[i] = p[min];
54          p[min] = temp;
55          if (p[i].at <= p[i - 1].ct) {
56              p[i].it = p[i - 1].ct;
57          } else {
58              p[i].it = p[i].at;
59          }
60          p[i].ct = p[i].it + p[i].bt;
61      }
62      printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
63      for (int i = 0; i < n; i++) {
64          p[i].tat = p[i].ct - p[i].at;
65          avgtat += p[i].tat;
66          p[i].wt = p[i].tat - p[i].bt;
67          avgwt += p[i].wt;
68          printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i].no, p[i].at, p[i].
    bt, p[i].ct, p[i].tat, p[i].wt);
69      }
70      avgtat /= n;
71      avgwt /= n;
72      printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f",
    avgtat, avgwt);
73      return 0;
74  }
```

Listing X.3: Código SJF

SRTF (Shortest Remaining Time First)

El algoritmo SRTF es una variante expropiativa del SJF, donde el proceso con el menor tiempo restante se ejecuta primero. Esto asegura un tiempo de espera promedio más bajo, sin embargo, puede generar problemas de inanición para procesos largos [117]. (Véase Tabla X.3)

Tabla X.3: Ejemplo del algoritmo SRTF [127]

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Paso a paso:

1. Llegada: Los procesos llegan en los tiempos 0, 1, 2 y 3.
2. Ejecución dinámica: Siempre se elige el proceso con el menor tiempo restante en cada instante.

Round Robin

El algoritmo Round Robin asigna a cada proceso un intervalo de tiempo fijo (*quantum*). Una vez que expira, el proceso se coloca al final de la cola [124]. (Véase Tabla X.4)

¿En qué se usa?

- Sistemas Operativos: En la planificación de CPU, para distribuir el tiempo de procesador equitativamente entre todos los procesos en ejecución, asegurando un tiempo de respuesta rápido y evitando el hambre de recursos [114].
- Redes de Computadoras: En la asignación de ancho de banda para garantizar que todos los nodos o clientes obtengan una porción justa del recurso, utilizado en algoritmos de planificación de tráfico y en la gestión de colas de paquetes [77].
- Sistemas de Tiempo Real: En donde se requiere que tareas críticas se ejecuten de manera periódica y predecible [62].

Tabla X.4: Ejemplo del algoritmo Round Robin (*Quantum* = 4) [52]

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	10
P2	1	4
P3	2	6

Paso a paso:

1. Los procesos llegan en los tiempos 0, 1 y 2.
2. Quantum fijo de 4 unidades:
   - P1 ejecuta 4 unidades ($t = 0$ a $t = 4$) y pasa al final de la cola.
   - P2 ejecuta sus 4 unidades completas ($t = 4$ a $t = 8$).
   - P3 ejecuta 4 unidades ($t = 8$ a $t = 12$) y pasa al final de la cola.
   - P1 reanuda y completa las 6 unidades restantes ($t = 12$ a $t = 18$).

Ejercicios resueltos

Implementar la planificación de procesos utilizando los algoritmos First-Come, First-Served (FCFS) y Shortest Job First (SJF), tomando como referencia la matriz presentada en la tabla X.5.

Tabla X.5: Ejercicio 1

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	12
P2	1	5
P3	2	9
P4	3	7
P5	4	3

A continuación, se muestra el ejercicio desarrollado manualmente. Los resultados se pueden apreciar en las Tablas: X.6, X.7.

- FCFS

Tabla X.6: Ejercicio 1 FCFS

Proceso	Arrival Time	(CPU Burst)	Start Time	Complete Time	Waiting Time	Turning Time
P1	0	12	0	12	0	12
P2	1	5	13	17	11	16
P3	2	9	18	26	15	24
P4	3	7	27	33	23	30
P5	4	3	34	36	29	32
				Aver. Waiting T. =	**15,6**	
					Aver. Tur. Time =	**22,8**

- SJF

Tabla X.7: Ejercicio 1 SJF

Proceso	Arrival Time	(CPU Burst)	Start Time	Complete Time	Waiting Time	Turning Time
P1	0	12	0	12	0	12
P5	4	3	13	15	8	11
P2	1	5	16	20	14	19
P4	3	7	21	27	17	24
P3	2	9	28	36	25	34
				Aver. Waiting T. =	**12,8**	
					Aver. Tur. Time =	**20**

Ejecución del ejercicio (Véase Figuras: X.1, X.2)

- FCFS

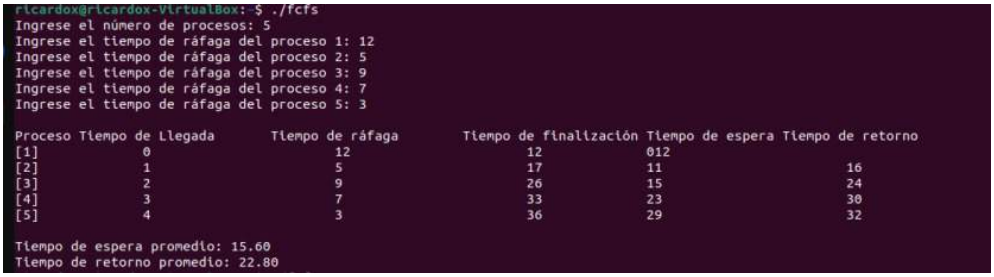


Figura X.1: Ejecución ejercicio 1 FCFS - SJF

- SJF

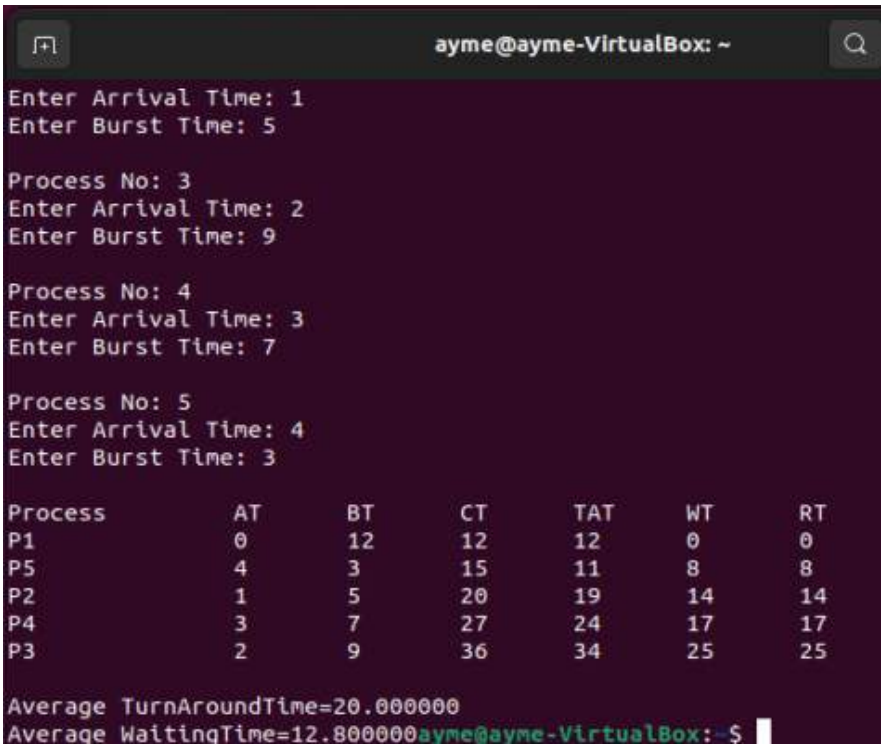


Figura X.2: Ejecución ejercicio 1 FCFS - SJF

Del mismo modo, se debe realizar la planificación de procesos empleando los algoritmos correspondientes en la Tabla X.8,

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	4
P2	1	3
P3	2	6
P4	3	5
P5	4	2
P6	5	4
P7	6	3
P8	7	5

Tabla X.8: Ejercicio 2

Ejercicio desarrollado manualmente (véase Tablas X.9, X.10)

- FCFS

Tabla X.9: Ejercicio 2 FCFS

Proceso	Arrival Time	(CPU Burst)	Start Time	Complete Time	Waiting Time	Turning Time
P1	0	4	0	4	0	4
P2	1	3	5	7	3	6
P3	2	6	8	13	5	11
P4	3	5	14	18	10	15
P5	4	2	19	20	14	16
P6	5	4	21	24	15	19
P7	6	3	25	27	18	21
P8	7	5	28	32	20	25
				Aver. Waiting T. =	**10,625**	
					Aver. Tur. Time =	**14,625**

- SJF

Tabla X.10: Ejercicio 2 SJF

Proceso	Arrival Time	(CPU Burst)	Start Time	Complete Time	Waiting Time	Turning Time
P1	0	4	0	4	0	4
P5	4	2	5	6	0	2
P2	1	3	7	9	5	8
P7	6	3	10	12	3	6
P6	5	4	13	16	7	11
P4	3	5	17	21	13	18
P8	7	5	22	26	14	19
P3	2	6	27	32	24	30
				Aver. Waiting T. =	**8,25**	
					Aver. Tur. Time =	**12,25**

Ejecución del ejercicio (Véase X.3, X.4).

- FCFS

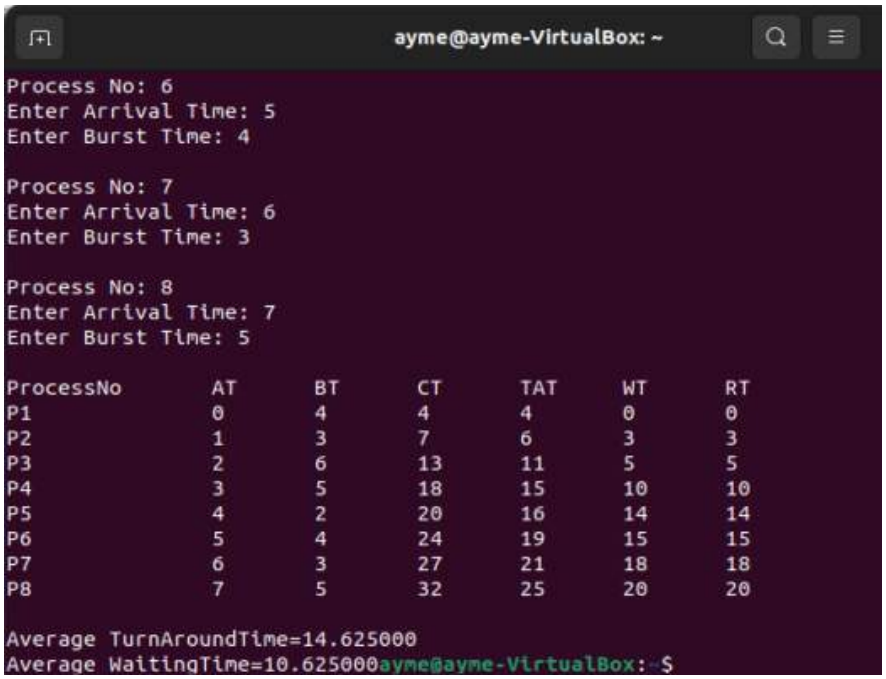


Figura X.3: Ejecución ejercicio 2 FCFS - SJF

- SJF

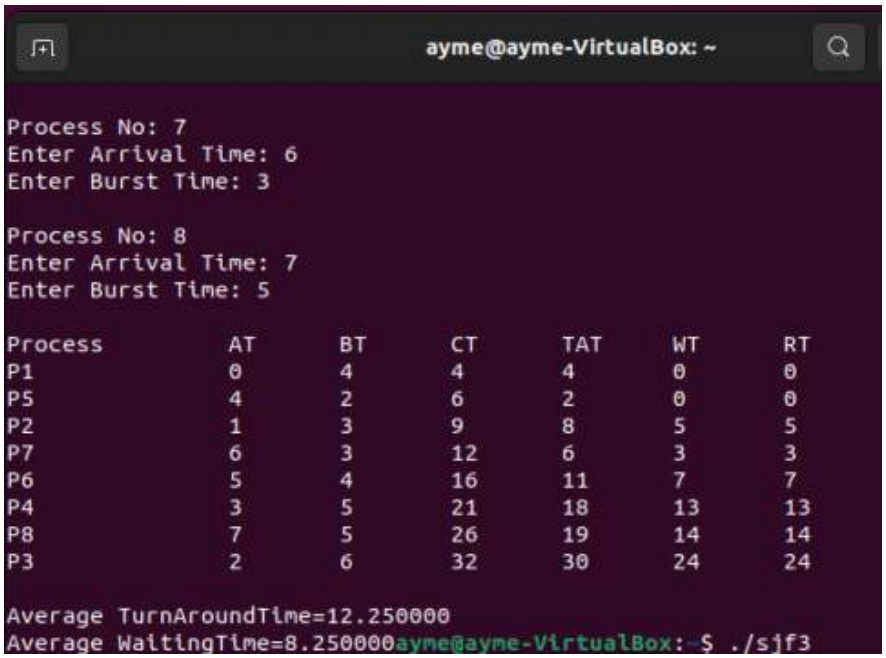


Figura X.4: Ejecución ejercicio 2 FCFS - SJF

Ejercicios propuestos

Ejercicio 1

Implementar la planificación de procesos utilizando los algoritmos First-Come, First-Served (FCFS) y Shortest Job First (SJF), tomando como referencia la matriz presentada en la tabla X.11.

Tabla X.11: Ejercicio 1

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	8
P2	2	4
P3	3	9
P4	5	6

Ejercicio 2

Realizar la planificación de procesos utilizando FCFS y SJF, considerando los datos de la tabla X.12.

Tabla X.12: Ejercicio 2

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	10
P2	1	5
P3	2	2
P4	3	7
P5	4	3

Ejercicio 3

Dado el conjunto de procesos en la tabla X.13, aplicar la planificación de procesos con FCFS y SJF.

Tabla X.13: Ejercicio 3

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	6
P2	1	8
P3	2	7
P4	4	3
P5	6	5
P6	8	4

Ejercicio 4

Utilizando los algoritmos FCFS y SJF, desarrollar la planificación de los procesos de la tabla X.14.

Tabla X.14: Ejercicio 4

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	3
P2	1	7
P3	2	2
P4	3	6
P5	5	4
P6	7	5

Ejercicio 5

Aplicar la planificación de procesos con FCFS y SJF utilizando los datos de la tabla X.15.

Tabla X.15: Ejercicio 5

Proceso	Tiempo de llegada	Duración (CPU Burst)
P1	0	5
P2	1	3
P3	2	9
P4	3	4
P5	4	6
P6	5	2
P7	6	8

Conclusiones del capítulo

La mayoría de los sistemas operativos tiene la propiedad de multiprogramación, lo que demanda de la planificación de procesos, en función de la prioridad, el algoritmo de asignación y los recursos disponibles. Cuando se dispone de algunos procesos en la cola de listos para ejecutarse, el sistema operativo debe decidir cuándo ejecutarlos.

El presente capítulo presentó los principales algoritmos de planificación de procesos como FCFS, SJF, Round Robin, así como varios parámetros, indicadores y métricas enfocadas a mejorar el rendimiento del sistema computacional como el tiempo de espera, el tiempo de respuesta, la penalización, el tiempo de retorno, el tiempo de completamiento, etc.

El algoritmo First-Come, First-Served (FCFS) ha sido analizado como el enfoque más simple, en el que los procesos se ejecutan en el orden en que llegan. Sin embargo, su estructura puede generar largos tiempos de espera cuando un proceso de gran duración se encuentra al inicio de la cola. En contraste, el algoritmo Shortest Job First (SJF) selecciona el proceso con menor tiempo de

ejecución, lo que reduce el tiempo de espera promedio. No obstante, puede causar que los procesos más largos esperen indefinidamente.

Lecciones aprendidas

- El tiempo total de respuesta es el tiempo obligatorio para terminar el proceso incluyendo el tiempo en estado de listo o preparado.
- El tiempo de espera es el tiempo en el cual un proceso pasa en la cola de preparados en estado de listo.
- El algoritmo FCFS es el más natural de implementar, pues considera los procesos en el orden de llegada a la cola de listos. Sin embargo, tiene la desventaja de que si un proceso tiene tiempos de espera largos, el rendimiento del sistema se verá seriamente afectado.
- Todos los algoritmos de planificación se basan en métricas e indicadores, lo cual permite medir y controlar.
- Un estudiante de ingeniería o un profesional técnico debe conocer los algoritmos de planificación de procesos que utiliza el sistema operativo.

Sitios de interés

Software de simulación de algoritmos de planificación de procesos.

Autor: Maté de Nicolás, Paulo, 2019. `https://e-archivo.uc3m.es/entities/publication/b5145c99-e22b-4a70-82c7-e7238b491059`

El planificador de procesos a través de un simulador.

Autores: Barrionuevo, Mercedes, Apolloni, Rubén, Piccoli, María Fabiana, 2009 `https://sedici.unlp.edu.ar/handle/10915/20945`

Simulador de Algoritmos del SO. Google Play.

Autor: Rafael López García. Última consulta: 06-01-2026. `https://play.google.com/store/apps/details?id=cs.rlopezga.osalgorithmsimulator&hl=es_EC`

Autoevaluación del capítulo

Responde las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿En qué consiste el overhead en un sistema operativo?**
   - Es una técnica del sistema operativo que optimiza las operaciones.
   - Es una penalización que soporta el usuario por la ejecución de varios procesos que requieren los recursos computacionales.
   - Técnica que sirve para disminuir la memoria virtual.
   - Método que facilita la creación de nuevos procesos.
2. **¿Cuál es una desventaja del algoritmo FCFS?**
   - No garantiza los procesos más cortos.
   - Puede generar tiempos de espera elevados si un proceso largo se encuentra al inicio de la cola.
   - Requiere conocimiento previo del tiempo de ejecución.
   - Solo funciona en sistemas Windows.
3. **¿En qué se diferencia el algoritmo SJF del FCFS?**
   - No ejecuta procesos en paralelo.
   - Es más fácil de implementar.
   - Selecciona el proceso con la menor duración de ráfaga.
   - Requiere que todos los procesos tengan la misma duración de ráfaga.
4. **¿Cuál es un problema potencial del algoritmo SJF?**
   - No se puede aplicar en sistemas operativos modernos.
   - Puede causar inanición de procesos largos si hay una llegada continua de procesos cortos.
   - No mejora el tiempo de espera promedio.
   - Solo se puede usar en sistemas monoprocesador.

5. **¿Qué métrica mide el tiempo total desde que un proceso entra en el sistema hasta que finaliza?**

   - Tiempo de espera.
   - Tiempo de retorno.
   - Tiempo de respuesta.
   - Uso del procesador.

6. **¿Cuál de las siguientes afirmaciones sobre los algoritmos de planificación es correcta?**

   - Cada algoritmo tiene ventajas y desventajas dependiendo del contexto de uso.
   - FCFS siempre es mejor que SJF.
   - La planificación de procesos solo es necesaria en sistemas multiprocesador.
   - No es posible medir la eficiencia de un algoritmo de planificación.

7. **¿Qué ocurre si un sistema operativo no usa un algoritmo de planificación eficiente?**

   - Los procesos se ejecutarán en menos tiempo.
   - Se evitarán bloqueos y tiempos de espera prolongados.
   - Se puede reducir la eficiencia del procesador y aumentar el tiempo de respuesta.
   - Los procesos siempre se ejecutarán en el orden correcto.

8. **¿Cuál de las siguientes estrategias de planificación es más justa en la distribución del tiempo de CPU?**

   - FCFS.
   - SJF.
   - Round Robin.
   - Ninguna de las anteriores.

CAPÍTULO XI

Gestión de Memoria Interna y de Almacenamiento

Introducción

La gestión de la memoria es una función fundamental de los sistemas operativos. La memoria es un elemento de un sistema computacional, dispositivo electrónico o de Internet de las cosas que permite almacenar temporalmente o físicamente los procesos en ejecución, instrucciones, datos e información. La memoria es uno de los recursos más valiosos a la hora de ejecutar las aplicaciones o servicios.

Sin embargo, a medida que es más requerida, es decir que se necesita de mayor capacidad computacional o de almacenamiento, el rendimiento de los referidos equipos disminuye.

La memoria ha experimentado un desarrollo interesante en los últimos tiempos, sea en capacidad, eficiencia y funcionamiento, lo que ha favorecido el desarrollo de sistemas de software, con mayor funcionalidad e innovación.

Este capítulo tiene como objetivo proveer a los lectores los conocimientos relacionados con la gestión de memoria, con el fin de fortalecer sus habilidades, destrezas, actitudes y competencias en administrar de forma técnica este importante recurso.

Este capítulo comprende los esenciales de la memoria interna o principal y de almacenamiento o secundaria, su estructura y funcionamiento. Además, serán analizadas las técnicas de gestión de memoria como la paginación, segmentación, de intercambio e híbrido.

Para fortalecer su aprendizaje se otorgarán varios comandos Linux tanto para gestión de memoria interna, como de almacenamiento, para comprender cómo gestionar la memoria en forma general.

Los animamos a leer y practicar este capítulo, con el fin de mejorar el rendimiento de este importante recurso.

¡Bienvenidos!

Esenciales de la memoria interna

La memoria interna también conocida como principal está activa mientras el computador está encendido. Sirve para almacenar temporalmente los datos, instrucciones o programas que están en ejecución.

Existen diferentes tipos, cada uno con sus diferentes características, tales como RAM, ROM, EPROM, FLASH, NVRAM, Caché. A continuación se caracteriza brevemente a algunas de ellas.

RAM (Random Access Memory)

RAM o memoria de acceso aleatorio. Es una memoria física temporal que está conectada directamente a las ranuras de la placa madre del computador [3]. La RAM recibe del procesador las instrucciones, programas o procesos en ejecución a fin de que sean almacenados temporalmente, y puedan ser sujetas a cálculos, organización, ordenamiento, lectura o ejecución.

La capacidad de memoria RAM depende de las necesidades de los usuarios y de su presupuesto. Su capacidad puede variar entre 8GB, 16GB, 32GB, 64GB o más. La capacidad de la RAM es un factor contundente ante los tipos de datos y la cantidad que se desee transmitir [84]. Así por ejemplo, si se requiere de grandes volúmenes de datos, cálculos matemáticos complejos, y mayor velocidad de transmisión, se requiere mayor capacidad. Mientras más capacidad de RAM se disponga, mayor costo se pagará. La Figura XI.1 muestra un ejemplo de una RAM reemplazable.

Figura XI.1: Memoria de Acceso Aleatorio reemplazable. **Fuente**: https://pixabay.com/es/photos/memoria-reemplazable-computadora-332492/

Las memorias RAM pueden utilizar módulos de dos tipos: DIMM (Dual In Line Module Memory) y la SODIMM (Small Outline Dual Inline Memory Module). Los DIMMs son los módulos de memoria en línea dual, y se los puede encontrar en equipos de escritorio y servidores. En cambio, los SODIMM son módulos de memoria en línea doble, que es una variante más compacta de DIMM estándar. Se utilizan principalmente en dispositivos donde el espacio es limitado, como laptops, tables y algunos mini PC

ROM (Read Only Memory)

Memoria de solo lectura, es un medio de almacenamiento no volátil o permanente utilizado en computadoras y dispositivos electrónicos, que permite solo la lectura de la información y no su escritura. Esta memoria se distingue porque no se pueden modificar una vez que se han grabado durante la fabricación, aunque existen otras versiones programables que se explicará posteriormente. Además, la información que se almacena en la ROM es permanente y no se destruye cuando se corta la energía eléctrica. Así por ejemplo, la BIOS, el firmware de dispositivos, controladores de hardware, y programas de arranque. Finalmente, la ROM se distingue por tener un acceso más lento en comparación con la RAM.

Entre los tipos de ROM, se tiene: Mask ROM (fija), PROM (programable una vez), EPROM (borrable con luz UV), EEPROM (eléctricamente programable y borrable), entre otras.

Comandos Linux para el monitoreo de memoria

A continuación, se describen algunos de los comandos que se utilizan para la gestión de memoria en Linux:

- **cat /proc/meminfo** Muestra la información de la memoria interna del archivo virtual.

  Comando que muestra la información de la memoria principal

  ```
  cat /proc/meminfo
  ```

- **free -h:** Proporciona información sobre la RAM y la memoria virtual.

Comando para visualizar información resumida de la RAM y de la memoria virtual

```
free -h
-b #Despliega en bytes.
-k #Despliega en Kbytes.
-m #Despliega en Mbytes.
-g #Despliega en Gbytes.
```

- **smem:** Permite desplegar un reporte del consumo de memoria a nivel de procesos, usuarios y en forma general del sistema vía consola.

Comando Linux que despliega el uso de memoria

```
smem
-u #Despliega despliega un reporte de consumo de
memoria por usuario.
-r #Despliega un reporte de mayor a menor.
-t #Despliega un reporte incluyendo la sumatoria
total de consumo.
-c #Selecciona las columnas para el despliegue
del reporte.
```

Comandos Linux para gestión de memoria de almacenamiento

A continuación, se describen algunos de los comandos más utilizados en la gestión de memoria externa o de almacenamiento en la terminal de Linux:

- **cat /proc/mount:** Permite visualizar los sistemas de archivos montados en el sistema en tiempo real.

Comando que muestra información en pantalla sobre todos los sistemas de archivos montados

```
cat /proc/mount
```

- **du (disk usage):** Sirve para visualizar el tamaño de archivos, subdirectorios y directorios.

Comando que muestra el espacio de almacenamiento

```
du
```

- **df (disk free):** Permite visualizar el espacio total, usado y libre en los sistemas de archivos montados.

Comando que muestra el espacio en disco disponible en el sistema de archivos

```
df -h
```

- **lsusb** Permite visualizar en pantalla la información sobre los dispositivos USB conectados al equipo, incluyendo el número de dispositivo, bus, ID de proveedor, ID de producto, el tipo de dispositivo, la subclase y el protocolo.

Comando que muestra todos los dispositivos USB conectados al equipo

```
lsusb
```

- **lspci** Permite visualizar en pantalla todos los componentes tipo Peripheral Component Interconnect (PCI) como tarjetas interfaz de red, tarjetas de sonido o de televisión.

Comando que muestra información sobre cada bus PCI del equipo

```
lsusb
```

- **lspci:** Permite visualizar en pantalla todos los componentes tipo Peripherical Component Interconnect (PCI) como tarjetas interfaz de red, de sonido o de televisión.

Comando que muestra información sobre cada bus PCI del equipo

```
lspci
```

- **lscpu:** Permite visualizar en pantalla todas las unidades de procesamiento (CPU).

Comando que muestra información sobre cada bus PCI del equipo

```
lscpu
```

- **lshw:** Permite visualizar en pantalla todas las unidades de procesamiento (CPU), memoria interna, de almacenamiento (disco), controladores, USB, interfaces de red, etc.

Comando que muestra información sobre cada bus PCI del equipo

```
lscpu
```

- **lsscsi:** Permite visualizar en pantalla todos los controladores SCSI / SATA, como los discos duros y las unidades ópticas.

Comando que muestra información sobre los dispositivos SCSI

```
lsscsi
```

- **fdisk:** Es un comando Liunux de particionado de discos.

Comando que permite formatear o particionar un disco duro

```
fdisk -l #Lista las particiones del disco.
fdisk -s #Muestra el tamaño asignado de una
participación.
fdisk -b #Especifica el tamaño del sector en KB,
MB, GB.
fdisk -v #Despliega la versión del Fdisk.
```

Técnicas de gestión de memoria

La gestión de memoria es un aspecto fundamental que influye en el rendimiento y la estabilidad de un sistema computacional. Los equipos tienen limitaciones en la memoria principal o interna, por lo tanto deben administrarse de manera eficiente. De esta manera, se garantizan que los procesos en ejecución tengan acceso a los recursos que sean necesario.

Para este propósito existen diversas técnicas de gestión de memoria, tales como Paginación, segmentación, técnicas híbridas, almacenamiento en caché, uso compartido de memoria e intercambio de memoria. Estas se utilizan para optimizar el uso de los recursos cuando están en ejecución los programas y procesos. En los siguientes apartados se describen brevemente estas técnicas, su funcionamiento, elementos principales, entre otros.

Paginación

De acuerdo con Iglesias [65] "La paginación es un esquema de gestión de memoria que divide la memoria física en bloques de tamaño fijo denominados marcos o páginas físicas. Además, divide el espacio de direcciones virtuales en bloques de tamaño fijo denominados páginas". La páginación toma la memoria y la particiona en páginas del mismo tamaño, otorgándole a cada proceso un espacio definido con los recursos necesarios o superiores, para su funcionamiento.

Entre los sistemas operativos que utilizan la paginación como técnica de gestión de memoria se pueden nombrar: Linux, Windows XP, Windows 11, Android, MacOS, Solaris, Windows NT, entre otros. Se muestra la lógica de esta técnica de memoria en la Figura XI.2 . A continuación, se describen las características de la Paginación:

- **División de la memoria en páginas y marcos:** La memoria lógica de un proceso se divide en páginas de tamaño fijo, mientras que la memoria física lo hace en marcos del mismo tamaño que las páginas.

- **Asignación no contigua:** No es necesario que un proceso ocupe un espacio continuo en la memoria física, estos pueden ubicarse en cualquier marco, optimizando el uso de la memoria.

- **Tabla de páginas:** Cada proceso tiene una tabla de páginas, estas traducen las direcciones lógicas de las físicas. Esta almacena un número de marco donde se encuentra cada página del proceso.

- **Evita la fragmentación externa:** Como estos bloques tienen un tamaño fijo, minimiza que espacios libres se desperdicien, Sin embargo, puede generar fragmentación interna si una página no usa todo el espacio del marco asignado.

- **Posibilidad de paginación por demanda:** No todas las páginas del proceso deben estar en memoria al mismo tiempo, estas se pueden cargar bajo demanda reduciendo el uso innecesario de memoria.

- **Uso de memoria virtual:** Combina la RAM con el almacenamiento secundario.

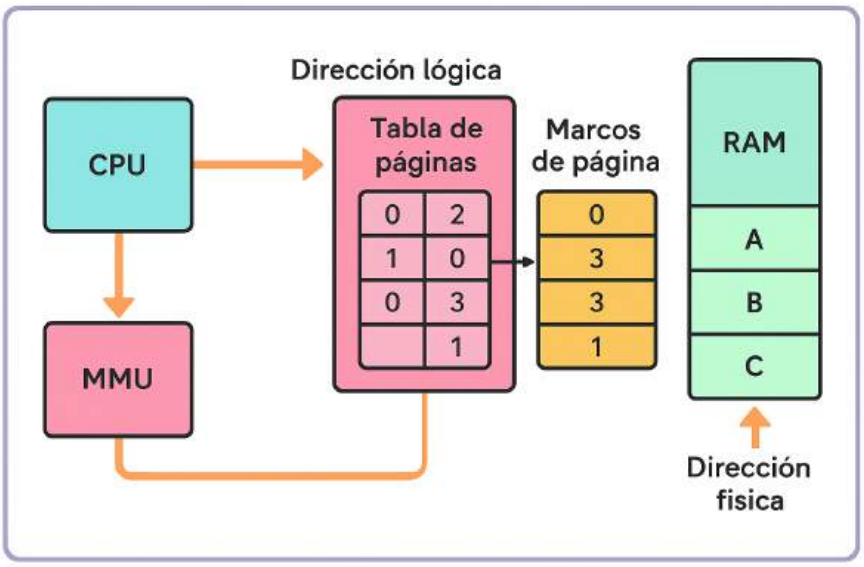


Figura XI.2: Diagrama Lógico de la Técnica de Paginación. Redibujado por Copilot GPT

Ejercicios

Ejercicio resuelto No. 1: Acceso a memoria virtual mediante paginación.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>

#define PAGE_SIZE 4096

int main() {
    void *page = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (page == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    snprintf((char *)page, "Hola, mundo de la paginaci n!");
    printf("%s\n", (char *)page);

    if (munmap(page, PAGE_SIZE) == -1) {
        perror("munmap");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Ejercicio resuelto No. 2: Asignar múltiples páginas y realizar operaciones de lectura y escritura en cada una.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>

#define NUM_PAGINAS 3
#define PAGE_SIZE 4096

int main() {
    void *paginas[NUM_PAGINAS];

    for (int i = 0; i < NUM_PAGINAS; i++) {
        paginas[i] = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE,
                          MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
        if (paginas[i] == MAP_FAILED) {
            perror("mmap");
            exit(EXIT_FAILURE);
        }
        sprintf((char *)paginas[i], "P gina %d", i);
    }

    for (int i = 0; i < NUM_PAGINAS; i++) {
        printf("Contenido de p gina %d: %s\n", i, (char *)paginas[i
    ]);
        munmap(paginas[i], PAGE_SIZE);
    }

    return 0;
}
```

Segmentación

La segmentación es la técnica de gestión de memoria que se basa en la división lógica del programa en partes denominadas segmentos. Cada segmento agrupa elementos relacionados lógicamente, como por ejemplo: la pila, código, datos, Heap (memoria dinámica) y otros valores. La segmentación permite un uso más eficiente de la memoria.

La segmentación consiste en dividir la memoria principal de una computadora en segmentos o secciones de diversos tamaños, conforme a la necesidad de ejecutar un proceso. Entre los sistemas operativos que utilizaban a la Segmentación se listan: OS/2 (IBM y Microsoft), MS-DOS, Windows 3.x / 9x. Como se puede apreciar, se trata de sistemas operativos antiguos, por lo que la Segmentación es una técnica antigua de gestión de memoria. La Figura XI.3 muestra su funcionamiento. A continuación se listan algunas características relevantes:

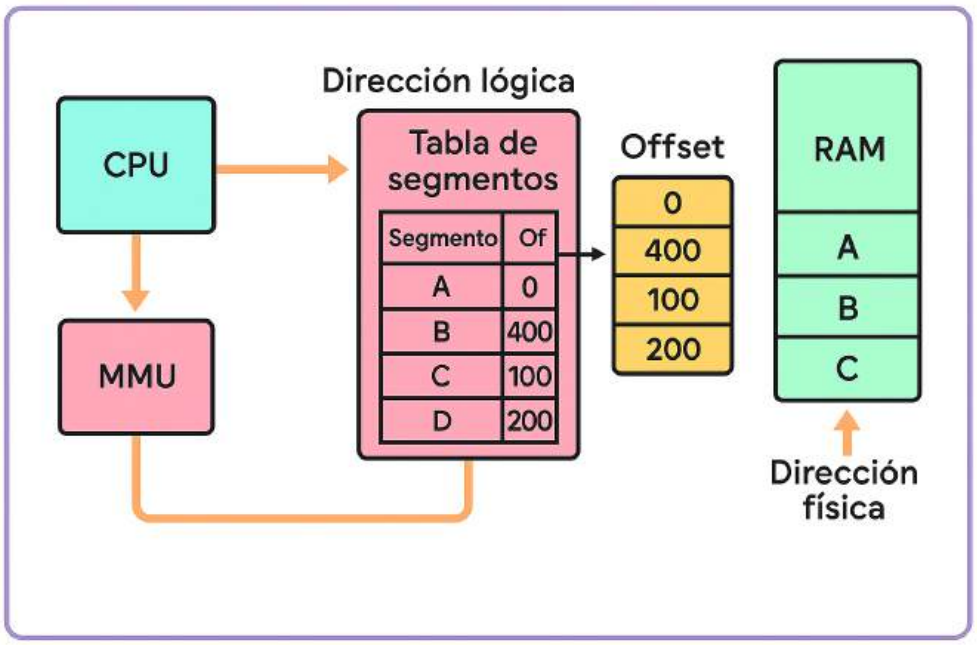


Figura XI.3: Diagrama Lógico de la Segmentación. Redibujado por Copilot GPT

Características

- **División de la memoria en segmentos:** Los procesos se dividen en segmentos lógicos de diferentes tamaños. La memoria física se asigna a cada segmento según su tamaño, esto dependerá del tipo de proceso: datos, pila, etc.
- **Asignación contigua por segmento:** Cada segmento se almacena en una parte de la memoria de manera continua, esto permite una mejor organización de los datos, y optimiza el uso de memoria.
- **Tabla de segmentos:** Todos los procesos tienen una tabla de segmentos, que es una estructura de datos para cada proceso, que guarda dirección base, y el límite de cada segmento. Se usa para conocer en dónde empieza la memoria física de ese segmento y para evitar accesos fuera de rango, respectivamente.
- **Traducción de direcciones mediante la unidad de gestión de memoria (MMU)**: Utiliza la tabla de segmentos para mapear y transformar entre direcciones lógicas y físicas.
- **Elimina la fragmentación interna:** La segmentación evita la fragmentación interna, puesto que los segmentos tienen tamaños variables, al

contrario con la paginación. Cabe señalar que la fragmentación interna ocurre cuando los bloques de memoria asignados contienen espacio no utilizado, que puede incidir negativamente en el rendimiento de un sistema y en la utilización de recursos.

- **Fragmentación externa:** Debido a que los segmentos pueden tener distintos tamaños y se almacenan en posiciones contiguas de la memoria, pueden generarse espacios libres entre ellos, lo que produce fragmentación externa.

Ejercicios

Ejercicio resuelto No. 2: Simulación de segmentación con estructuras.

```
#include <stdio.h>

struct Segmento {
    int base;
    int limite;
};

int main() {
    struct Segmento codigo = {0, 100};
    struct Segmento datos = {100, 200};
    struct Segmento pila = {200, 300};

    printf("Segmento de c digo: Base = %d, L mite = %d\n", codigo.
    base, codigo.limite);
    printf("Segmento de datos: Base = %d, L mite = %d\n", datos.
    base, datos.limite);
    printf("Segmento de pila: Base = %d, L mite = %d\n", pila.base,
     pila.limite);

    return 0;
}
```

Ejercicio resuelto No. 2: Verificar si una dirección pertenece a un segmento dado.

```
#include <stdio.h>

struct Segmento {
    int base;
    int limite;
};

int pertenece(struct Segmento seg, int direccion) {
    return direccion >= seg.base && direccion < seg.limite;
}

int main() {
    struct Segmento datos = {100, 200};
```

```
    int direccion = 150;

    if (pertenece(datos, direccion)) {
        printf("La direcci n %d pertenece al segmento.\n",
   direccion);
    } else {
        printf("La direcci n %d NO pertenece al segmento.\n",
   direccion);
    }

    return 0;
}
```

Híbrida

La técnica de gestión de memoria híbrida combina la paginación junto con la segmentación, aprovechando sus ventajas y disminuyendo sus desventajas. Esto permite que los procesos sean más efectivos, mejorando el rendimiento de los mismos. A continuación, se analizan sus características principales.

- **Combinación de Técnicas:** Se puede combinar segmentación con paginación, de esta manera se obtiene un sistema eficiente. Con ello se logra evitar la fragmentación externa gracias a la paginación, y mejorar la organización lógica del programa con la segmentación.
- **Optimización del Uso de Memoria:** Asigna solo el espacio necesario, reduce el desperdicio de memoria, y en consecuencia, mejora el rendimiento.
- **Traducción de Direcciones Compleja:** Todos los procesos tienen una tabla de segmentos, esta guarda dirección base, y el límite de cada segmento.

Ejercicios

Ejercicio resuelto No. 3: Segmentación paginada.

```
#include <stdio.h>
#include <stdlib.h>

#define NUM_SEGMENTOS 4
#define PAGINAS_POR_SEGMENTO 8
#define TAMANO_PAGINA 4096

struct TablaPaginas {
    int paginas[PAGINAS_POR_SEGMENTO];
};

```

```c
struct TablaSegmentos {
    struct TablaPaginas *segmentos[NUM_SEGMENTOS];
};

int main() {
    struct TablaSegmentos ts;

    for (int i = 0; i < NUM_SEGMENTOS; i++) {
        ts.segmentos[i] = malloc(sizeof(struct TablaPaginas));
        for (int j = 0; j < PAGINAS_POR_SEGMENTO; j++) {
            ts.segmentos[i]->paginas[j] = i * PAGINAS_POR_SEGMENTO + j;
        }
    }

    int segmento = 2;
    int pagina = 5;
    int direccion_fisica = ts.segmentos[segmento]->paginas[pagina] * TAMANO_PAGINA;
    printf("Direcci n  f sica: %d\n", direccion_fisica);

    for (int i = 0; i < NUM_SEGMENTOS; i++) {
        free(ts.segmentos[i]);
    }

    return 0;
}
```

Ejercicio resuelto No. 2: Manejar fallos de página simulados y carga desde almacenamiento secundario.

```c
#include <stdio.h>
#include <stdlib.h>

int cargar_pagina(int segmento, int pagina) {
    printf("Cargando p gina  %d del segmento %d desde almacenamiento ...\n", pagina, segmento);
    return segmento * 100 + pagina;
}

int main() {
    int segmento = 1, pagina = 3;
    int direccion = cargar_pagina(segmento, pagina) * 4096;
    printf("Direcci n  f sica cargada: %d\n", direccion);
    return 0;
}
```

Ejercicios propuestos

Ejercicio 1

¿Qué ventajas posee la técnica de gestión de memoria Paginación vs la Segmentación? Justifique su respuesta

Ejercicio 2

¿Qué función tiene la MMU?

Ejercicio 3

¿La memoria virtual también es conocida como la memoria de intercambio?

Ejercicio 4

Se tiene un proceso en memoria virtual, cuyo espacio lógico de direcciones contiene 300 páginas de 256 KB cada una. Además, se conoce que el tamaño de la memoria es de 32 MB. ¿Cuál es la cantidad de bytes que componen la dirección virtual y la dirección física? Justifique su respuesta.

Ejercicio 5

¿Cuáles son los servicios de gestión de memoria del sistema operativo Linux?

Conclusiones del capítulo

Este capítulo ha proporcionado una introducción práctica y conceptual de la gestión de memoria, que es un recurso determinante en la eficiencia de los sistemas computacionales. Los lectores han aprendido cómo la memoria se clasifica, cuáles son sus técnicas de gestión y cómo se realiza la gestión de memoria en Linux. Se han explorado conceptos esenciales de la memoria interna. Además, mediante comandos Linux se ha aprendido cómo gestionar la memoria interna. Comandos Linux para gestión de memoria de almacenamiento. Técnicas de gestión de memoria como la Paginación, Segmentación caracterizándolo, incluyendo ejercicios resueltos y ejercicios propuestos. Los ejemplos prácticos y problemas resueltos proporcionados a lo largo del capítulo han permitido consolidar estos conocimientos, ayudando al lector a desarrollar habilidades para gestionar la memoria interna o principal como la de almacenamiento. Al finalizar este capítulo, el lector no solo comprende los fundamentos de la gestión de memoria, sino que además aprende a caracterizarla y a distinguirla.

Lecciones aprendidas

- La memoria ha experimentado un desarrollo importante en los últimos tiempos, sea en capacidad, eficiencia y funcionamiento, lo que ha favorecido el desarrollo de sistemas de software, con mayor funcionalidad e innovación.
- La memoria interna también conocida como principal está activa mientras el computador está encendido. Sirve para almacenar temporalmente los datos, instrucciones o programas que están en ejecución.
- La gestión de memoria es un aspecto fundamental que influye en el rendimiento y la estabilidad de un sistema computacional.
- La paginación es un esquema de gestión de memoria que divide la memoria física en bloques de tamaño fijo denominados marcos o páginas físicas.
- La segmentación consiste en dividir la memoria principal de una computadora en segmentos o secciones de diversos tamaños, conforme a la necesidad de ejecutar un proceso.

Sitios de interés

La empresa **Samsung Electronics** `https://semiconductor.samsung.com/about-us/business-area/memory/`) es la compañía líder mundial tanto en memoria interna como de almacenamiento. En menor capacidad se incluyen SK hynix `https://product.skhynix.com/products/dram/dram.go`) y Micron `https://www.micron.com/products/memory/dram-components`) ,que completan el trío de gigantes que fabrican memoria a nivel global.

Stephen St. Michael, **Introduction to DRAM (Dynamic Random-Access Memory**, August 01, 2019. URL:`https://www.allaboutcircuits.com/technical-articles/`

SSD University, **Comprehensive Guide to SSDs: Advantages & Key Features**, 2024-01-31. URL: `https://www.ssstc.com/knowledge-detail/guide-to-ssds-advantages-and-features/`

Autoevaluación del capítulo

Responda las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿Cuál es el acrónimo de las Siglas DRAM?**
   - Data Recognition Access Memory.
   - Dynamic Random Access Memory
   - Direct Random Access Memory.
   - Deducible Random Access Memory.
2. **Otorgue el comando Linux necesario para desplegar en pantalla el archivo virtual que contiene información variada sobre el uso de la memoria en tiempo real**
   - top.
   - $ cat /proc/meminfo.
   - htop.
   - vmstat.
3. **¿Cuál es la función de la MMU?**
   - Transforma una dirección virtual a física.
   - Prioriza los procesos con menor memoria.
   - Asigna mayor cantidad de memoria al proceso necesitado.
   - Alterna memoria entre procesos sin seguir un orden específico.
4. **¿Encuentre las razones por la que memoria principal y la memoria de almacenamiento son diferentes?**
   - La memoria principal es aquella que almacena el sistema operativo.
   - La memoria principal es temporal mientras está encendido el computador. La de almacenamiento es permanente, almacena datos, archivos, bases de datos, y las librerías de los sistemas operativos.
   - La memoria principal tiene mayor capacidad que la de almacenamiento.

- Los HDD y los SSD son memoria externas de almacenamiento y tienen mayor precio de la memoria interna.

5. **¿La memoria de acceso aleatorio (RAM) difiere de la memoria de solo lectura (ROM) debido a?**

   - La ROM tiene empotrado programas de Software.
   - Los datos e información se mantienen en la RAM pese a que se apague el sistema computacional.
   - La ROM es una memoria solo de lectura, mientras que la RAM es de acceso aleatorio.
   - La RAM está más cerca de la memoria Caché L1, L2 y L3.

6. **¿En qué consiste la técnica de gestión de memoria Paginación?**

   - Consiste en dividir la memoria principal de una computadora en segmentos o secciones de diversos tamaños, conforme a la necesidad de ejecutar un proceso.
   - Es un esquema de gestión de memoria que divide la memoria física en bloques de tamaño fijo denominados marcos o páginas físicas.
   - Se trata de una técnica de gestión de memoria que representa las necesidades reales de un programador.
   - Permite que las partes de un proceso se coloquen en frames de memoria física contiguos.

7. **¿En qué consiste la técnica de gestión de memoria Segmentación?**

   - Es un esquema de gestión de memoria que divide la memoria física en bloques de tamaño fijo denominados marcos o páginas físicas
   - La segmentación es una técnica de administración de memoria que divide dinámicamente la memoria en diferentes segmentos. Cada segmento tiene un tamaño que se ajusta a lo que visión del programador, por tanto, puede tener un tamaño diferente.
   - Permite que las partes de un proceso se coloquen en frames de memoria física no contiguos.
   - Divide la memoria lógica como la física en bloques de igual tamaño.

8. **Indique el resultado del siguiente comando: fdisk -l**

- Despliega por pantalla las particiones del disco duro.
- Despliega por pantalla las particiones con controlador SCSI.
- Lista todos los dispositivos del sistema.
- Visualiza el nombre del host.

9. **¿Defina disco en estado sólido (Solid State Drive, SSD)?**

   - Utilizan magnetismo para almacenar los datos.
   - Utiliza platos magnéticos concéntricos, sectores y pistas.
   - Disco que utiliza memoria no volátil, como la memoria flash, para almacenar datos.
   - Es una memoria de almacenamiento de acceso directo.

10. **¿La paginación es una técnica de administración de memoria que funciona de la siguiente manera:**

    - Es una técnica de administración de memoria que divide dinámicamente la memoria en diferentes segmentos. Cada segmento tiene un tamaño que se ajusta a lo que visión del programador, por tanto, puede tener un tamaño diferente.
    - Divide la memoria lógica como la física en bloques de diferente tamaño.
    - Divide la memoria física en bloques de tamaño fijo llamados frames (marcos de página) y a la memoria lógica en bloques del mismo tamaño llamados pages (páginas).
    - Ninguna de las anteriores.

CAPÍTULO XII

Gestión de Servicios de Redes en Linux

Introducción

Los sistemas operativos son responsables de la configuración de servicios en redes de computadores [30]. Estos servicios permiten a los usuarios (i.e., clientes, operadores, o empresarios) acceder a sus aplicaciones que residen en el ciberespacio. Algunos ejemplos comunes son el servidor Web (HTTP), el correo electrónico (SMTP, POP3, IMAP4), la traducción de nombres de dominio (DNS), el acceso remoto seguro (SSH), el servidor DHCP, entre otros.

Para levantar servicios en redes, los sistemas operativos facilitan la funcionalidad de configuración de tarjetas de red, direccionamiento IP, enrutamiento, gestión puertos, de firewalls, y servicios de redes. [130].

Dentro de este contexto, el sistema operativo Linux dispone de comandos, herramientas y funcionalidades de configuración y operación de servicios de red a través del kernel, que incluye la gestión de funciones de networking e internetworking, la comunicación con el hardware subyacente y la gestión de procesos [41]. Conocerlas permitirá al lector construir y operar redes Linux y levantar servicios, conforme a sus necesidades.

Este capítulo aporta en los lectores información teórica y práctica para que conozcan cómo se instalan y configuran los servicios de networking en una plataforma Linux.

Al finalizar este capítulo, el lector comprenderá los conceptos básicos de redes de computadores basados en Linux, incluyendo varios comandos y herramientas de gestión de redes, levantamiento de servicios, direccionamiento IP, enrutamiento, etcétera. Así mismo, realizará varias pruebas de conectividad y verificará si los servicios estarán debidamente instalados. Finalmente, aprenderá a realizar el monitoreo de rendimiento de las redes.

En este capítulo el lector también encontrará actividades de aprendizaje, programas propuestos, resueltos, y sitios de interés que facilitarán la apropiación de su aprendizaje.

¡Bienvenidos!

Servicios de redes Linux

Los servicios de redes en Linux son aplicaciones o procesos que permiten la interconexión y comunicación entre diferentes sistemas. Estos servicios se gestionan a través de herramientas y comandos específicos, que facilitan tanto la configuración de la conectividad como la administración de usuarios [1]. Entre los servicios más comunes que se pueden instalar, configurar y poner en operación, se listan a continuación con una leve descripción:

- **Servicio Web**: responsable de poner en funcionamiento el protocolo HTTP, HTTPS, para conectarse con sitios, páginas o portales web que muestren la información, o aplicaciones de los clientes o de las empresas.

- **El Servicio de correo electrónico**: permite acceder vía una interfaz web a la mensajería necesaria para distribuir la información o comunicación entre los actores de una empresa.

- **El servicio SSH**: que es un servicio de acceso remoto seguro que facilita las comunicaciones encriptadas a distancia para precautelar la confidencialidad, integridad y disponibilidad de la información.

- **El servicio DNS**: que es un sistema distribuido, responsable de realizar la traducción de nombres de dominio hacia una dirección IP o viceversa.

- **El Servicio DHCP**: que consiste en asignar automáticamente un rango de direcciones IP en una red puesta en operación.

Topología de experimentación y pruebas basada en Linux

La figura XII.1, muestra un entorno virtual de red controlado basado en Linux que será utilizado para una explicación práctica de algunos de los capítulos de este libro. Por ejemplo, cuando se prueben varios comandos de conectividad en Linux, se levantan servicios de red en el servidor virtual o cuando se realizan ataques simulados a redes IP, según como corresponda en sus respectivos capítulos.

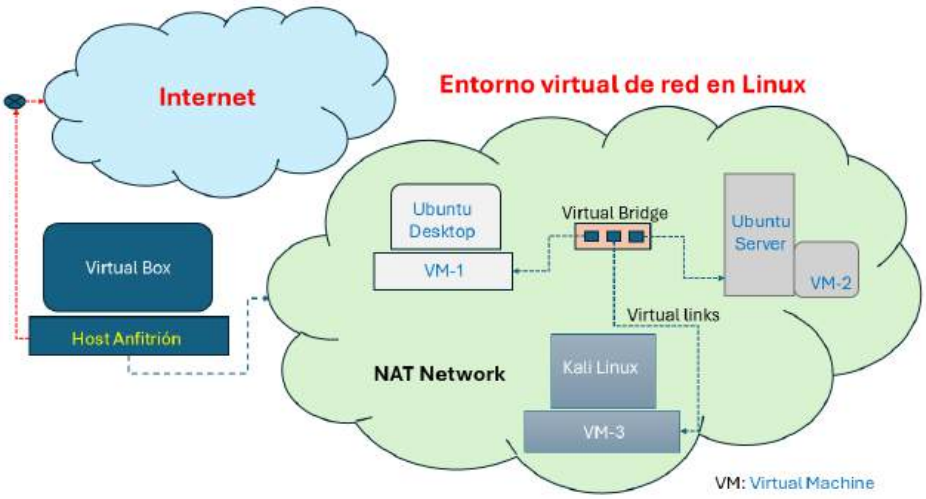


Figura XII.1: Abstracción visual de un entorno virtual de red para experimentación y pruebas de networking en Linux

Tal como se puede apreciar, a partir de un equipo físico o *host anfitrión*, se configuró un *entorno virtual de red en Linux* para las pruebas [49]. En este caso se seleccionó la plataforma de Virtualización *Virtual Box*. En el equipo anfitrión configuró una máquina virtual en la que se instaló Ubuntu para escritorio *(VM-1)*, otra con un sistema operativo de servidor *(VM-2)*. Finalmente se instaló y otra máquina virtual con Kali Linux *(VM-3)*, que, en este caso, sería la maquina atacante.

Para la conectividad, Virtual Box permite configurar una *red NAT* [103], la cual permite traducir direcciones IP hacia el Internet utilizando un *bridge virtual* que se conecta con la interfaz de red del host anfitrión. De esa manera, todas las máquinas virtuales pueden disponer de una tarjeta virtual de red y sus respectivos *enlaces virtuales*, con una dirección de hardware virtual (MAC), a la que se le asigna una dirección IP que le permite ser un nodo conectado al Internet.

Este entorno virtual de red fue configurado completamente con software de código abierto y de libre distribución, reúne las ventajas de disminuir los costos de inversión de hardware y software, reduce costos de experimentación, administración, mantenimiento y de electricidad [50]. En realidad, se trata de un entorno completamente configurable, seguro y escalable que será utilizado en el desarrollo de este libro.

Comandos Linux para gestión de conectividad en Linux

Para verificar y configurar la conectividad en un sistema Linux se utilizan diversos comandos, entre los cuales destacan:

- **`ifconfig`**: Aunque en muchas distribuciones ha sido reemplazado por el comando `ip`, `ifconfig` sigue siendo ampliamente utilizado para mostrar y modificar la configuración de las interfaces de red.

  Comando que muestra información en pantalla sobre las direcciones IP de las diferentes interfaces de red disponibles en el equipo

  ```
  $ ifconfig
  ```

- **`ip`**: Es la herramienta moderna para la administración de redes en Linux. Permite gestionar interfaces, rutas, direcciones IP y más.

  Comando para gestión de redes

  ```
  $ ip
  ```

- **`ping`**: Utilizado para comprobar la conectividad con otro equipo enviando paquetes de datos y midiendo la latencia.

  Comando para verificar conectividad de red

  ```
  $ ping -c 5 IP
  ```

- **`netstat`** o **`ss`**: Sirve para visualizar las conexiones de red activas, las rutas y la utilización de puertos.

  Comando para ver conexiones de red

  ```
  $ netstat -inetd
  ```

- **`route`**: Permiten visualizar la tabla de enrutamiento a la cual ese equipo está conectado física e inalámbricamente.

  El comando route muestra o modifica la tabla de enrutamiento IP.

  ```
  $ route -n
  ```

- **`nslookup`**: Es un comando Linux utilizado para obtener información del servidor del sistema de nombres de dominio (Domain Name System, DNS).

Permite consultar al servidor DNS para obtener la traducción de la asignación de nombres de dominio o direcciones IP o cualquier otro registro DNS específico. También existen dos opciones adicionales:
$ dig
$ host

```
$ nslookup
```

- **mtr**: Es un comando Linux de diagnóstico de red y solución de problemas que combina ping y traceroute en una sola herramienta.

El comando MTR proporciona datos en tiempo real sobre el retardo de la red; la pérdida de paquetes y los problemas de enrutamiento.

```
$ mtr -4b www.google.com
```

- **iperf**: Es una herramienta Linux que permite ejecutar pruebas de velocidad localmente sin necesidad de conexión a Internet.

Iperf permite mediciones activas de la capacidad de transmisión máxima.

```
$ iperf -c 10.0.2.4
```

- **hostname**: Es un comando Linux que permite cambiar o actualizar el nombre del equipo o nodo de la red.

Hostname otorga un nombre al equipo o dispositivo el cual es reconocido por los restantes equipos conectados a esa red.

```
$ hostname alfa
```

Comandos Linux para gestión de cuentas de usuario

La administración de usuarios es vital para controlar el acceso a los recursos y servicios del sistema. Los siguientes comandos son los principales en la gestión de cuentas de usuario:

- **useradd**: Crea nuevas cuentas de usuario.
- **usermod**: Se utiliza para modificar atributos de cuentas de usuarios existentes (nombres, directorios home, shells, grupos, bloqueos de seguridad).

- **`passwd`**: Se utiliza para establecer o cambiar la contraseña de un usuario.
- **`deluser`**: Remueve de forma definitiva del directorio /home/ la cuenta del usuario especificado.

Configuración de servicios de Redes en Distribuciones Linux

Los servicios de las redes de computadoras basadas en Linux permiten que los usuarios puedan acceder a los diferentes servicios y aplicaciones en red. Así, por ejemplo, los usuarios necesitan tener acceso remoto seguro, por lo cual deben instalar el servicio **SSH**. Acceder a sus cuentas de correo electrónico, por lo tanto, deberán configurar los servicios de **email** y **DNS**. En el siguiente apartado se explorarán algunos de estos servicios.

SSH (Secure Shell)

SSH es un protocolo de red que permite establecer una conexión remota segura entre dos nodos para la administración remota y la transferencia segura de archivos [28]. Funciona cifrando la comunicación entre los dos equipos, lo que lo asegura la transferencia de los datos. Entre sus principales características se pueden señalar:

- **Cifrado de extremo a extremo**: La comunicación entre cliente y servidor está cifrada, protegiendo el sistema contra accesos no autorizados
- **Autenticación robusta**: El servicio incluye múltiples métodos de autenticación incluyendo contraseñas, claves públicas, privadas y certificados digitales.
- **Integridad de datos**: Utiliza códigos de autenticación de mensajes para asegurar que los datos no han sido modificados durante la transmisión
- **Compresión de datos**: Capacidad de comprimir datos antes de encriptarlos para optimizar la capacidad de transmisión.
- **Túneles seguros**: Permite crear túneles cifrados para proteger otros protocolos inseguros (port forwarding)
- **Transferencia segura de archivos**: Incluye protocolos SCP y SFTP para la copia y transferencia encriptada de archivos

- **Multiplexación de canales**: Permite múltiples sesiones simultáneas sobre una única conexión SSH.

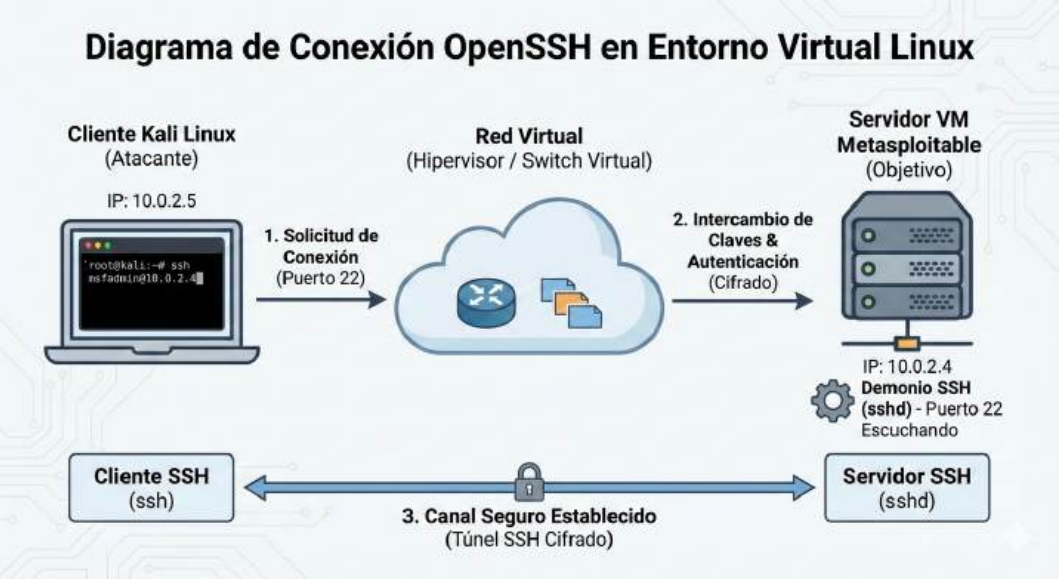
La antigua versión de SSH, ha sido sustituida en los últimos años por OpenSSH. En las distribuciones de Linux el nombre del paquete para el servidor es openssh-server y para el cliente es openssh-client [93]. A continuación se describen brevemente sus principales elementos a ser configurados:

- **openssh-server**: Demonio del servidor SSH (sshd) que analiza las conexiones entrantes en el puerto 22
- **openssh-client**: Cliente SSH que permite conectarse a servidores remotos.
- **Puerto predeterminado**: 22/TCP
- **Protocolo**: SSH-2 (versión actual)
- **Algoritmos de cifrado**: RSA, AES, ChaCha20, 3DES, se utilizan para proteger la información transmitida.
- **Intercambio de claves**: Métodos como Diffie-Hellman, ECDH.
- **Archivos de configuración**:
  - `/etc/ssh/sshd_config`: Parámetros del servidor SSH.
  - `/etc/ssh/ssh_config`: Configuración del cliente SSH.
  - `~/.ssh/`: Directorio del usuario donde se guardan las claves y configuraciones personales.
- **Estados del servicio**
  - **En ejecución (running)**: El servicio está activo y funcionando.
  - **Finalizado (dead)**: El servicio se encuentra detenido.
  - **Falló (failed)**: El servicio presenta un error al iniciar.
  - **Activado (enabled)**: El servicio está configurado para iniciar automáticamente con el sistema.
  - **Desabilitado (disabled)**: El servicio no se inicia automáticamente.

SSH emplea el puerto 22. La autenticación la realiza mediante la clave de usuario y su contraseña, que consiste en dos claves encriptadas asimétricamente. Permite un cifrado punto-a-punto, tunelización segura, y es una solución que resuelve los problemas de los protocolos de transferencia de archivos FTP (File Transfer Protocol) que ha quedado en obsolescencia.

Para el propósito de este libro, para simular la conexión remota segura mediante SSH, se empleará un entorno virtual de red controlado basado en Linux, con Kali Linux como cliente y una VM Metasploitable como servidor. La Figura XII.2 muestra sus elementos y el funcionamiento.

Figura XII.2: Topología de experimentación de la conexión remota segura mediante SSH. **Fuente:** Ilustración generada por los autores del libro, utilizando Gemini IA Gen de `https://www.Google.com.\`



El proceso para poner en funcionamiento el servicio SSH en una distribución de Linux es el siguiente:

- Instalar el servicio SSH en un servidor Linux mediante el gestor de paquetes apt (i.e., advanced packet tool), verificando la correcta instalación de los componentes openssh-server y openssh-client.
- Configurar y Verificar el estado del servicio SSH utilizando systemctl, en conocimiento de los comandos de inicio, suspensión, reinicio y verificación del estado del servicio.
- Abrir el puerto y activarlo en el firewall.
- Crear las claves RSA tanto en el cliente como en el servidor.

- Verificar el funcionamiento del servicio SSH con la lectura de los logs del sistema y el ingreso a través del puerto 22.
- Realizar varias copias seguras desde el cliente hacia el servidor y viceversa mediante scp.

A continuación, en la Tabla XII.1 se muestra un resumen de los comandos utilizados desde el proceso de instalación, configuración de claves, conexión remota, copias de seguridad, y puesta en ejecución:

Comando	Acción
`$sudo apt update`	Actualiza el índice de paquetes disponibles para descarga
`apt install openssh-server`	Instala el servidor SSH
`systemctl status ssh`	Muestra el estado actual del servicio
`systemctl start ssh`	Inicia el servicio SSH
`systemctl stop ssh`	Detiene el servicio SSH
`systemctl restart ssh`	Reinicia el servicio SSH
`sudo ufw allow ssh`	Habilita el puerto 22 del servicio SSH
`sudo ufw enable`	Activa el servicio SSH en el firewall
`sudo ufw status`	Permite verificar el estado del firewall
`systemctl enable ssh`	Habilita inicio automático al arranque
`systemctl disable ssh`	Deshabilita inicio automático
`systemctl is-enabled ssh`	Verifica si está habilitado
`ssh-keygen -t rsa -b 2048`	Genera las claves RSA con una longitud 2048 bits
`ssh-X root@10.0.2.4`	Accede a la interfaz de la VM Metasploitable2
`scp file.txt`	Copia el archivo file.txt a la VM Metasploitable2

Tabla XII.1: Resumen de comandos para la instalación, configuración y puesta en funcionamiento de SSH

Servicio Web

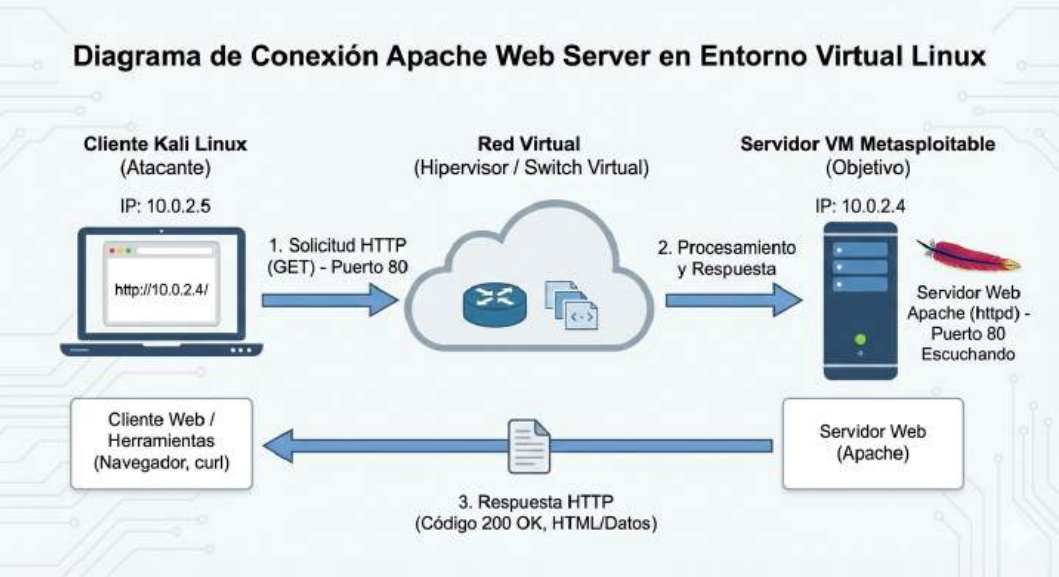
El servidor web de código abierto y de libre distribución más utilizado en el mundo es Apache Web Server [69]. Es desarrollado y respaldado por la Fundación de Software Apache (Apache Software Foundation, ASF) en `www.https://www.apache.org/` [19].

En el sector de las TI, existe un conjunto de tecnologías de software de código abierto conocidas como LAMP (Linux, Apache, MySQL y PHP) que permiten alojar y gestionar el contenido de sitios web y aplicaciones web dinámicas [21]. Estos elementos se explican brevemente: Linux es el sistema operativo que sustenta toda la tecnología. Apache es el servidor web que gestiona las solicitudes HTTP entrantes de los navegadores y sirve el contenido web.

MySQL se utiliza para gestionar bases de datos relacionales. PHP es un lenguaje de scripting del lado del servidor que permite procesar contenido dinámico, comunicarse con bases de datos y generar HTML que se envía al navegador del cliente. Todos estos elementos funcionan conjuntamente como un sistema cohesivo.

Apache es un software de servidor web gratuito, de código abierto y multiplataforma [69]. Se encarga de gestionar las solicitudes de acceso de los clientes a una dirección web. Apache gestiona estas solicitudes, que llegan mediante el protocolo HTTP, y responde a los clientes. Apache puede enviar y recibir todo tipo de archivos, incluyendo imágenes, PDF y otros archivos descargables a través del navegador [60]. Además, puede ejecutar código en lenguajes de programación específicos, lo que permite a los desarrolladores crear páginas mediante programación del lado del servidor. Apache también actúa como intérprete e intermediario entre el navegador del usuario y el contenido del sitio web. Su eficiencia y fiabilidad se basan en su arquitectura modular y su capacidad para gestionar múltiples solicitudes simultáneamente [68]. En la Figura Figura XII.3 se muestra sus elementos y el funcionamiento.

Figura XII.3: Topología de experimentación del Servidor Apache Web en Linux. **Fuente:** Ilustración generada por los autores del libro, utilizando Gemini IA Gen de `https://www.Google.com.\`



En la carpeta /etc del árbol de directorios de Linux se encuentran los archivos de configuración del servidor Web Apache. Específicamente en la dirección: /var/www/html/ en donde suele estar ubicado el archivo index.html que contiene la página Web de arranque del servicio. A continuación se listan

los archivos de configuración del servicio:

- **/etc/apache2/**: Carpeta principal de configuración
- **/etc/apache2/apache2.conf**: Archivo que contiene la configuración del servicio.
- **/etc/apache2/sites-available/**: Sitios web disponibles
- **/etc/apache2/sites-enabled/**: Sitios activos (enlaces simbólicos)
- **/etc/apache2/mods-available/**: Módulos disponibles
- **/etc/apache2/mods-enabled/**:Módulos activos

El servidor web Apache también almacena registros que contienen información detallada sobre cada solicitud que recibe, incluyendo la dirección IP del cliente, la URL solicitada, la fecha de la solicitud y la acción realizada. También registra cualquier error o anomalía. Los dos tipos principales son los registros de acceso, que incluyen detalles de las solicitudes, y los registros de errores, que registran problemas o advertencias emitidas por el servicio. Los registros del servidor se enumeran a continuación:

```
/var/log/apache2/access.log         # Log de accesos
/var/log/apache2/error.log          # Log de errores
```

El proceso para poner en funcionamiento el servicio Apache Web en una distribución de Linux es el siguiente:

- Instalar el servicio Apache web en un servidor Linux mediante el gestor de paquetes apt, verificando la correcta instalación de los componentes sudo apt install apache2.
- Configurar y verificar el estado del servicio HTTP utilizando systemctl, en conocimiento de los comandos de inicio, suspensión, reinicio y verificación del estado del servicio sudo systemctl status apache2.
- Habilitar el puerto del servicio y activarlo en el firewall mediante el ufw. Activar el puerto 80 para tráfico web normal y no cifrado, el puerto 443 para tráfico cifrado con TLS y SSL, o solo el puerto 443 para tráfico cifrado sudo ufw allow apache2
- Verificar y validar el correcto funcionamiento del servicio apache2 mediante la revisión de logs del sistema y el estado del puerto 80 o 443.

- Realizar varias descargas desde el cliente hacia el servidor y viceversa mediante wget.

A continuación en la Tabla XII.2 se muestra un resumen de los comandos utilizados desde el proceso de instalación, configuración, activación del firewall, descarga de archivos, y puesta en ejecución:

Comando	Acción
`$sudo apt update`	Actualiza el índice de paquetes disponibles para descarga
`apt install apache2`	Instala el servidor Apache Web
`systemctl status apache2`	Muestra el estado actual del servicio
`systemctl start apache2`	Inicia el servicio Apache Web
`systemctl stop apache2`	Detiene el servicio Apache Web
`systemctl restart apache2`	Reinicia el servicio Apache Web
`ufw allow apache2`	Habilita el puerto 80 del servicio Apache Web tradiciional
`ufw allow 'Apache Full'`	Habilita el puerto HTTP (80) y HTTPS (443)
`ufw allow 'Apache Secure'`	Habilita solo el puerto 443
`sudo ufw status`	Permite verificar el estado del firewall
`systemctl enable apache2`	Habilita inicio automático al arranque
`systemctl disable apache2`	Deshabilita inicio automático
`systemctl is-enabled apache2`	Verifica si está habilitado
`systemctl reload apache2`	Recarga la configuración sin detener
`wget https://file.zip`	Descarga de archivos según el URL establecido

Tabla XII.2: Resumen de comandos para la instalación, configuración y puesta en funcionamiento de Apache Web Server

Ejercicios resueltos

En esta sección conviene revisar ejercicios resueltos, especialmente de comandos Linux para gestión de servicios.

1. Otorgue el comando Linux para verificar el estado del servicio `SSH`:

Comando para verificar el estado del servicio SSH

```
$ sudo systemctl status SSH
```

2. Otorgue el comando Linux arrancar el servicio `Apache Web`:

Comando para arrancar el servicio Apache Web

```
$ sudo systemctl start apache2
```

3. Otorgue el comando Linux para visualizar por pantalla las reglas activas del `firewall de Linux`:

Comando para visualizer las reglas activas del firewall

```
$ sudo ufw status
```

4. Otorgue el comando Linux para eliminar la regla del puerto 80 del `firewall de Linux`:

Comando para desactivar el Puerto 80

```
$ sudo ufw delete allow 80/tcp
```

5. Otorgue el comando Linux para instalar el `cliente OpenSSH`:

Comando para instalar el cliente SSH

```
$ sudo apt install openssh-client
```

6. Otorgue el comando Linux para instalar en Linux el servicio `Web`:

Comando para instalar el servicio Web en Linux

```
$ sudo apt install apache2
```

7. Otorgue el comando Linux para instalar el `servidor OpenSSH`:

Comando para instalar el servicio SSH

```
$ sudo apt install openssh-server
```

Ejercicios propuestos

Ejercicio 1

Investigue cómo instalar el servicio de correo electrónico en Linux. Para este fin, existen en la industria dos servidores en Linux, Postfix y Sendmail. Sendmail fue precursora en la entrega de correos en sistemas Unix y Linux. Postfix, en cambio, se ha convertido en la opción actual. Luego instale y configure en la VM Metasploitable2 Posix.

Ejercicio 2

Liste los archivos de configuración del servidor de correo electrónico

Ejercicio 3

Liste los comandos Linux necesarios para instalar el servicio de correo electrónico en Linux

Ejercicio 4

Liste los comandos necesarios para activar todos los perfiles del servidor Apache Web, incluyendo Apache Full, y Apache Secure.

Ejercicio 5

Elabore una matriz que permita determinar con argumentos las diferencias entre Posfix y Sendmail.

Conclusiones del capítulo y lecciones aprendidas

Este capítulo ha proporcionado una visión general dos importantes servicios de redes Linux. El servidor SSH es el responsable de otorgar una comunicación remota segura encriptada extremo a extremo. Por su parte, el servidor Web es el responsable de la transferencia de información entre páginas y sitios Web. La instalación y configuración de servicios en redes son tareas que se las realiza empleando los sistemas operativos. Por lo tanto, es esencial que el lector aprenda sobre como instalarlos y configurarlos. Los ejemplos prácticos y problemas resueltos proporcionados a lo largo del capítulo han permitido consolidar estos conocimientos, ayudando al lector a desarrollar habilidades para gestionar servicios de redes entornos virtuales controlados. Al finalizar

este capítulo, el lector no solo comprende los fundamentos de los servicios en redes, sino que además comprende como instalarlos y configurarlos.

Lecciones aprendidas

- Para levantar servicios en redes, los sistemas operativos facilitan la funcionalidad de configuración de tarjetas de red, direccionamiento IP, enrutamiento, gestión puertos, de firewalls, y servicios de redes.
- El servicio SSH es un servicio de acceso remoto seguro que facilita las comunicaciones encriptadas a distancia para precautelar la confidencialidad, integridad y disponibilidad de la información.
- SSH emplea el puerto 22. La autenticación la realiza mediante la clave de usuario y su contraseña, que consiste en dos claves encriptadas asimétricamente
- Apache es un software de servidor web gratuito, de código abierto y multiplataforma. Se encarga de gestionar las solicitudes de acceso de los clientes a una dirección web.

Sitios de interés

Sitio Web oficial de la Fundación de Software Apache (ASF) `https://www.apache.org`, en donde se muestran sus productos y servicios. La ASF es una organización sin fines de lucro, creada en 1999 como patrocinador del Servidor Web Apache y para apoyar proyectos de software de código abierto y libre distribución.

Sitio web oficial de RedesZone, `www.https://www.redeszone.net/`, en donde se pueden encontrar una variedad de manuales y tutoriales sobre comandos de redes Linux.

Sitio web oficial de Apuntes de Programador, `https://apuntes.de/linux-certificacion-lpi/#gsc.tab=0`, que reúne información técnica de aspectos relacionados con Linux, tanto básico como avanzado, direccionamiento IP, instalación y configuración de servicios, desde el punto de vista de un programador.

Autoevaluación del capítulo

Responda las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿Qué significa HTTPS?**
   - Hyper Text Transmission Protocol Secure.
   - Hyper Text Transfer Protocol Secure
   - Hyper Text Transmision Protocol Security.
   - Ninguna de las anteriores.
2. **Defina Apache Web Server**
   - Es un software de servidor Web comercial
   - Es un software de servidor web gratuito y de código abierto para plataformas Unix y Linux cuyo patrocinador es la fundación de Software Apache.
   - Es un servidor que sostiene que los servicios y datos estén accesibles cuando se necesiten por parte de los usuarios.
   - Es la posibilidad de transmitir hipertexto en diferentes páginas y sitios web.
3. **¿Cuál es la función de un servidor SSH?**
   - Todas las anteriores
   - Permite una conexión remota encriptada segura extremo a extremo.
   - Debe ser instalado tanto en el cliente como en el servidor.
   - Utiliza el puerto 22 por defecto.
4. **¿Cuál es la diferencia entre un cliente y un servidor SSH?**
   - No existe ninguna diferencia, los dos funcionan igual.
   - El cliente que realiza solicitudes para acceder a servicios web, el servidor que atiende dichas solicitudes.
   - El cliente es un browser para establecer una conexión con cualquier servidor Web, disponga o no de perfiles de acceso.

- El cliente es el encargado de la comunicación fluida y segura con el servidor.

5. **¿Cuáles son los elementos de un localizador uniforme de recursos (URL)?**
   - No siempre están definidos.
   - El protocolo o servicio, el nombre del servidor, el sistema de nombres de dominio asociado, el archivo index.html.
   - Depende del protocolo de comunicaciones utilizado.
   - El protocolo de comunicación, el nombre del servidor físico, el nombre geográfico, el sitio web

6. **¿Cuáles son los elementos de un identificador uniforme de recursos, URI?**
   - No siempre están definidos
   - El esquema que lo identifica, el nombre del host que contiene el servicio, un path y una cadena de consulta
   - Depende del protocolo de comunicaciones utilizado.
   - El protocolo o servicio, el nombre del servidor, el sistema de nombres de dominio asociado, el archivo index.html.

7. **Defina servidor de nombres de dominio**
   - Es un software que permite que las direcciones IP se traduzcan y asocien a los nombres de dominio que ha sido configurado para acceder a una página Web.
   - Es el responsable de atender las solicitudes de un browser y ejecutar lo que este requiera.
   - Es aquel encargado de gestionar los correos electrónicos entrantes y salientes.
   - Ninguna respuesta es correcta.

8. **¿Cuál es el conjunto de comandos para instalar el servicio SSH en Linux?**
   - Depende de la versión.

- Se debe instalar solo en el servidor y no en el cliente.
- sudo apt install openssh-client, sudo apt install openssh-server, keygen-ssh -t rsa, sudo systemctl status SSH, sudo ufw allow 22/tcp.
- Ninguna de las anteriores.

CAPÍTULO XIII

Gestión de ciberseguridad en Sistemas Operativos

Introducción

De acuerdo con la ISO 32000 [106], la Ciberseguridad es una parte de la Seguridad de la Información, encargada de proteger la seguridad de las redes de computadoras, los dispositivos de networking, las redes de telefonía celular, el Internet y las infraestructuras de misión crítica de los estados. Consiste en varios mecanismos, estándares internacionales, procedimientos, métodos y técnicas, generalmente aceptados para minimizar las amenazas y vulnerabilidades cibernéticas.

La ciberseguridad en sistemas Linux es esencial para la protección de los datos que circulan por las redes y que son almacenados en servidores, portátiles y otros dispositivos de networking. Las distribuciones de Linux prometen ser aquellas que otorgan mayores niveles de seguridad puesto que son las menos atacadas y porque disponen de un robusto sistema de acceso y permisos de archivos y directorios. No obstante, en contra podrían ser vulnerables, si no son debidamente configurados y si no se actualizan sus librerías y parches de seguridad [87].

En concreto, los ambientes Linux también tienen diversas amenazas y vulnerabilidades que podrían poner en riesgo la triada CID [139]. Por lo tanto, estimado lector, es necesario el aprendizaje de los conceptos fundamentales de ciberseguridad y la forma de configurar los mecanismos para mitigarlos como control de acceso, monitoreo de tráfico y rendimiento, que mejoran la seguridad en ambientes de red basados en Linux [82], [51].

Este capítulo tiene como propósito, dar al lector los fundamentos esenciales de ciberseguridad y los mecanismos para configurar algunos niveles de seguridad utilizando el sistema operativo Linux.

En el desarrollo del capítulo usted encontrará tanto el contenido teórico como práctico para su mejor comprensión. Además, se presentan varios ejercicios resueltos y propuestos para la aplicabilidad práctica.

¡Bienvenidos!

Objetivos de seguridad en redes Linux

A continuación, se listan los objetivos de seguridad de Linux, que son coincidentes y convergen con los estándares de ciberseguridad más populares, tales como ISO 27000, NIST Cybersecurity Framework y CIS Controls [97]:

- **Confidencialidad:** Asegura que tanto la información transmitida como sus claves de acceso, sean secretas y no difundidas. En Linux, se logra mediante permisos de archivos, cifrado y protocolos seguros como SSH, TLS, SSL, entre otros [86].
- **Integridad:** Significa que la información no puede ser alterada o modificada de forma no autorizada. Esto se asegura con funciones hash tipo SHA-256, firmas o certificados digitales y mecanismos de verificación de integridad de paquetes como AIDE [109].
- **Disponibilidad:** Sostiene que los servicios y datos estén accesibles cuando se necesiten en un 99.999 % por parte de los usuarios. Linux utiliza técnicas como la redundancia, balanceo de carga, respaldos y medidas contra ataques DDoS [56].
- **Autenticación:** Es un factor de seguridad que sostiene que el usuario que se autentica debe demostrar que es él. En Linux se utilizan técnicas como Hash y otras técnicas multifactor para prevenir la autenticación.
- **Trazabilidad:** Es la posibilidad de verificar el histórico de las acciones preliminares que se produjeron en un evento, y que constituye evidencia para recuperar un proceso o sistema. En Linux se utilizan técnicas de auditoría, el histórico de comandos utilizados, control de integridad, control de versiones, entre otras.

Mecanismos de seguridad en redes Linux

- **Inteligencia de amenazas:**

  Es una tendencia para identificar, analizar y anticipar posibles amenazas y vulnerabilidades que podrían comprometer la triada CID y además la autenticación, trazabilidad, no repudio y resiliencia de los recursos administrados por el sistema operativo. Su propósito es anticipar ataques cibernéticos, detectar y mitigarlos antes de que dichas amenazas sean

cristalizadas [105]. Se caracteriza por ser proactiva, sistémica, y colaborativa. Para lograrlo, utiliza diferentes mecanismos tales como el análisis de tráfico malicioso, el control de acceso basado en roles (RBAC) y las listas de control de acceso (ACL) en el sistema de archivos. Del mismo modo, emplea herramientas como OWASP Threat Dragon y MITRE ATT&CK Navigator [64]. Finalmente, emplea metodologías de modelado de amenazas como STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), útiles para identificar amenazas específicas y proteger el sistema operativo [111].

- **Gestión de vulnerabilidades:**

  La gestión de vulnerabilidades es un proceso crítico y permanente para reducir las amenazas y en consecuencia los riesgos. Este proceso se basa en las primeras fases del Hacking Ético, tales como el reconocimiento pasivo, activo y el escaneado. El fin es identificar antes que el delincuente, cibernético puertos abiertos o en estado de escucha, versiones de los servicios instalados o aplicaciones obsoletas, la carencia de parches en los sistemas operativos, y evaluar las debilidades de seguridad que pueden ser explotadas por los ciber atacantes [53]. Para este fin existen diversos mecanismos como por ejemplo el escaneo de puertos [10] y las herramientas especializadas para este fin como *Nmap, Nikto, OpenVas*. Del mismo modo, la distribución de *Kali-Linux* posee el framework *Metasploit* el mismo que reúne varias opciones para análisis de vulnerabilidades y los exploits para detectar, evaluar, priorizar y mitigarlas posibles ataques cibernéticos. [120].

- **Pruebas de penetración** Son una práctica de ciberseguridad relacionada con el Hacking Ético, en la que personas o empresas autorizadas simulan ciberataques en sistemas computacionales para identificar vulnerabilidades antes de que sean explotados por los atacantes cibernéticos. Este proceso se inicia con un reconocimiento pasivo, activo, el escaneo de vulnerabilidades, la explotación, post explotación y limpieza de rastros son el fin de incrementar la seguridad de la información de una empresa [113].

- **Criptografía:**

  Hoy en día, dada la creciente amenaza de los ataques cibernéticos en frecuencia y complejidad, es fundamental el uso de la Criptografía en el

ámbito de los sistemas operativos porque se integran con sus servicios de autenticación, confidencialidad, integridad y no repudio [75]. La Criptografía proviene de dos términos griegos: cripto, oculto y grafos, escritura.

La Criptografía es una disciplina que estudia la forma de cifrar un mensaje para que no sea interceptado y descifrado por un delincuente cibernético (criptoanálisis) [57]. La Criptografía y el Criptoanálisis son subconjuntos de la Criptología que es la ciencia encargada de proteger los datos por medio de métodos, técnicas y algoritmos para que solo los destinatarios autorizados puedan acceder a ella.

Al llegar a su destino, el mensaje puede ser descifrado de modo que solo los usuarios autorizados puedan leer el mensaje original. La Criptografía puede ser simétrica, lo que implica que el remitente y el receptor pueden leer el mensaje utilizando una única llave privada. También puede ser asimétrica, lo que significa que se requieren de dos claves, una pública y otra privada para leer el mensaje [101]. El transmisor utiliza la clave pública del receptor para cifrar el mensaje. El receptor utiliza su clave privada para descifrar el mensaje.

Finalmente, para disponer de sistemas más seguros de comunicación, la Criptografía, puede ser combinada con una función Hash, que es una técnica y un algoritmo matemático. Hash es el resultado de aplicar una función matemática a un archivo, mensaje, contraseña, etc., y que proporciona una cadena alfanumérica única, irreversible y determinista que la representa e identifica [108]. Sus principales funciones es verificar tanto la integridad de los datos, pues cualquier modificación altera el Hash, así como la autenticidad en los sistemas de firma digital.

Firewalls en Linux y su configuración

Los firewalls en Linux son mecanismos tradicionales para filtrar y controlar el tráfico que entra y sale de la red. Consiste en un conjunto de políticas o reglas configuradas por el administrador (root) con el propósito de proteger el sistema computacional [42]. En Linux existen varias soluciones como las siguientes:

- **iptables:** Mecanismo clásico y mínimo para configurar reglas de filtrado de paquetes en el sistema operativo. Permite controlar qué puertos están abiertos, qué tipo de paquetes se intercambian y que direcciones IP tienen permiso de acceso, y bajo qué condiciones.

Ejemplo: Para bloquear todo el tráfico entrante en el puerto 80 (HTTP), utilice:

Bloqueo de todo el tráfico entrante en el puerto 80

$ sudo iptables -A INPUT -p tcp –dport 80 -j DROP
En dónde:

- *-A INPUT*: añade una regla a la cadena de entrada.
- *-p tcp*: especifica el protocolo TCP.
- *–dport 80*: indica el puerto de destino.
- *-j DROP*: descarta el tráfico sin notificar.

Otros ejemplos básicos con iptables:

Configuración de iptables:

```
sudo iptables -A INPUT -p tcp -dport 22 -j
ACCEPT
sudo iptables -A INPUT -p tcp -dport 80 -j
ACCEPT
sudo iptables -P INPUT DROP
```

- **ufw (Uncomplicated Firewall):** Interfaz simplificada para iptables que facilita la asignación de las reglas del firewall. Se requieren privilegios de administrador. Es común en distribuciones como Ubuntu, Debian y Kali Linux.

Ejemplo: Para bloquear el puerto 80 (HTTP), utilice:

Bloqueo de todo el tráfico entrante en el puerto 80 mediante ufw

$ sudo ufw deny 80/tcp
En dónde:

- *deny*: permite denegar el tráfico.
- *80/tcp*: especifica el puerto y el protocolo.

Otros ejemplos básicos con ufw:

- *$ sudo ufw status*: permite visualizar por pantalla el estado del firewall de Linux.
- *$ sudo ufw allow ssh*: permite activar el puerto y protocolo de la conexión remota segura.
- *$ sudo ufw allow from 192.168.100.101*: Otorga permiso de acceso a cualquier puerto o protocolo a la dirección IP 192.168.100.101.

Funciones Hash

Las funciones hash son algoritmos matemáticos que transforman una entrada de datos en una cadena de longitud fija, representando la huella digital del contenido. Es un valor alfanumérico único que se genera a partir de un mensaje, un dato, o un archivo. En seguridad de la información, además se utilizan para la creación de firmas digitales muy útiles para firmar digitalmente un documento.

Funciones de hash típicas:

- *MD5*: Algoritmo criptográfico de resumen de mensajes.
- *SHA-1*: Algoritmo de hashing más seguro.
- *SHA-256*: Recomendada actualmente por su robustez muy utilizada en criptomonedas.
- *SHA2-354*: Recomendada para certificados digitales, TLS/SSL y aplicaciones Web.
- *SHA2-512* Recomendada para certificados digitales TLS/SSL, aplicaciones Web, criptomonedas y Blockchain.

- *SHA3*: Recomendada para resistencia contra ataques de estructura, sistemas criptográficos, hashing y Blockchain. [8].

Aplicación en Ubuntu desktop:

```
sha256sum fichero1.txt
```

Aplicación en la terminal de Linux

$ echo "La ESPE es una universidad emblemática» fichero1.txt
Se visualizará una representación alfanumérica resultante con diferentes algoritmos:
$ md5sum fichero1.txt
$ sha1sum fichero1.txt
$ sha256sum fichero1.txt
$ sha3sum fichero1.txt

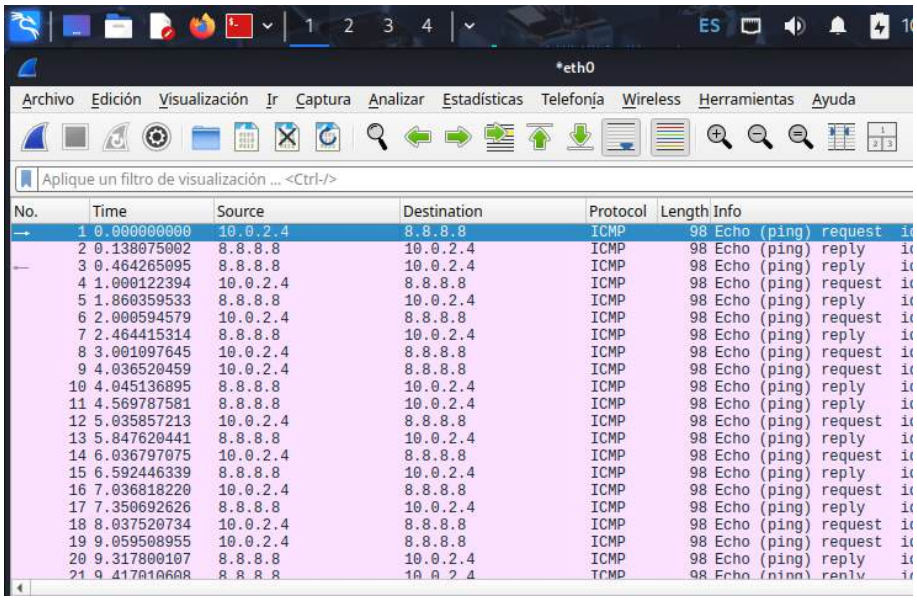
La respuesta será diferente en cada algoritmo.

Monitoreo y análisis de tráfico en redes Linux

El monitoreo de tráfico y protocolos es una actividad permanente del administrador de la red o del usuario de manera general para mantener servicios de calidad, realizar seguimiento, diagnosticar fallas y resolver problemas. Consiste en analizar el flujo de paquetes que se transmiten en la red con el fin de medir el desempeño o rendimiento de la red, detectar amenazas de seguridad y monitorear el uso de recursos [47] (ver Figura. XIII.1).

La Figura XIII.1 muestra el análisis de tráfico mediante Wireshark [89], obtenida en un escenario virtual controlado con Kali Linux. Como se puede apreciar, se trata de tráfico ICMP, en donde se visualiza por pantalla el número de paquete, tiempo entre llegada de los mismos, dirección IP origen, dirección IP destino, el protocolo ICMP, la longitud del paquete y la información del mismo.

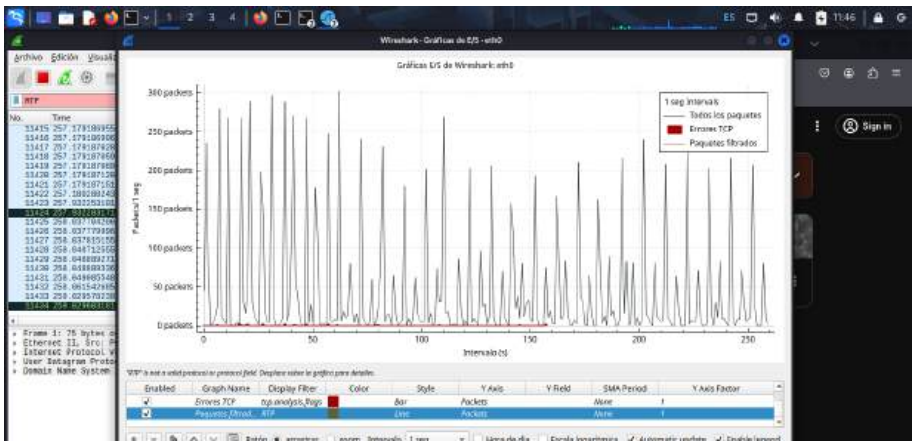
Figura XIII.1: Monitoreo de tráfico utilizando Wireshark en Kali Linux.



El monitoreo de rendimiento de la red

Evalúa el desempeño de general de la red. Dentro de sus métricas e indicadores se incluye el retardo (jitter), el retardo solo de una vía (one way delay), el troughtput, la capacidad de transmisión, el tiempo entre llegada de paquetes, el tiempo de respuesta, tiempo de espera, tiempo de retorno (Ver Figura XIII.2).

Figura XIII.2: Estadísticas de tráfico en la red utilizando Wireshark en Kali Linux.



La Figura XIII.2 muestra los picos y variaciones típicas del streaming de video de 50 paquetes en intervalos de 1 segundo- milisegundos.

Ataques simulados a redes Linux

Ataques de escaneo de puertos con Nmap

Nmap **(Netwotk Mapping)** es una herramienta central para el mapeo de la superficie de ataque a un equipo computacional y a la red en la que está conectado [43]. El entender el procedimiento del ataque y conocer cómo es detectada/bloqueada también es relevante [80].

Un ataque de escaneo de puertos significa utilizar un programa de software para realizar análisis de vulnerabilidades. El atacante intenta encontrar los puertos abiertos, las versiones de los servicios, protocolos o aplicaciones que estén en estado de abierto en el equipo víctima para detectar vulnerabilidades en ese equipo y evitar que un atacante pueda infiltrarse. En esta obra utilizaremos la herramienta tradicional denominado nmap.

Comando que permite escanear una PC en base a una Dirección IP

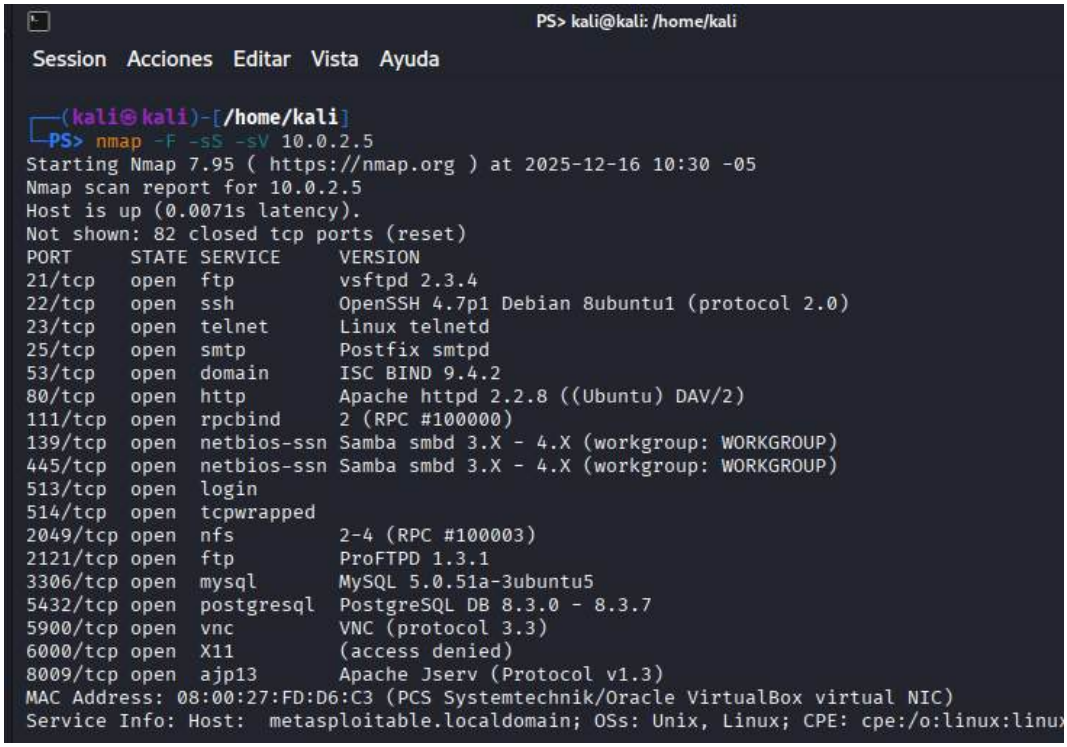
```
nmap -sn 192.168.10.101
#Descubrir si el host está activo.
nmap -T4 -F 192.168.10.101 -on nmap_rapido.txt
#Escaneo de puertos TCP (rápido)
nmap -p- -sS -sV -O -reason -open 10.0.2.4 -oA
nmap_full
# Escaneo profundo con detección de versiones y SO
nmap -script=banner -p 21,22,80,139,445 10.0.2.15
-oN nmap_banner.txt
# Banner y metadatos
nmap -script=smb-enum-shares,smb-enum-users -p
139,445 10.0.2.4 -oN nmap_smb.txt
#Bannerymetadatos
#SMB(comparticiones, usuariosannimos)
```

La Figura XIII.3 muestra un ejemplo de un escaneo de puertos (i.e., reconocimiento activo, escaneo, y enumeración), en las fases de Ethical Hacking. Como se puede observar, se trata de un ataque en un entono virtual controlado, cuyo atacante es una máquina virtual con Kali Linux y la víctima es la VM Metasploitable 2. Se trata de un escaneo rápido (opción -F), un escaneo medio abierto enviando paquetes TCP SYN a los puertos abiertos (opción -sS) y un escaneo de versiones de las aplicaciones y servicios (opción -sV).

La Figura XIII.3 además muestra los diferentes puertos, el estado abierto (open) o de escucha (listening), la versión de servicios o protocolos la MAC address del equipo escaneado, así como el tipo de sistema operativo y su versión. Con esta información los atacantes pueden realizar la explotación y post-

explotación como parte de las fases de Ethical Hacking.

Figura XIII.3: Escaneo de puertos utilizando nmap.



Ataques DDoS con hping3

hping3 es una herramienta de código abierto en línea de Linux que tiene funciones similares al ping tradicional. Por ejemplo, permite verificar que exista conectividad, mostrando la ruta que siguen los paquetes desde su origen hasta su destino en Internet, visualizando los saltos intermedios, sus tiempos de respuesta, la latencia, paquetes enviados, recibidos y perdidos. La Figura XIII.4 presenta la aplicación de hping3

Para la siguiente simulación de ataques de redes IP, en un entorno virtual controlado, se utilizará la misma topología de prueba. Kali Linux será el atacante y la VM Metasploitable 2 la máquina virtual víctima junto con hping3. La Figura XIII.5 muestra como a través de esta herramienta, se pueden inyectar un volumen masivo de paquetes ICMP, TCP, UDP e inundar progresivamente el equipo víctima, hasta que colapse. Para mayor información se recomienda acceder al sitio Web oficial de Kali Linux: `https://www.kali.org/tools/hping3/`

Figura XIII.4: Funcionalidades de hping3: traceroute en Kali Linux.

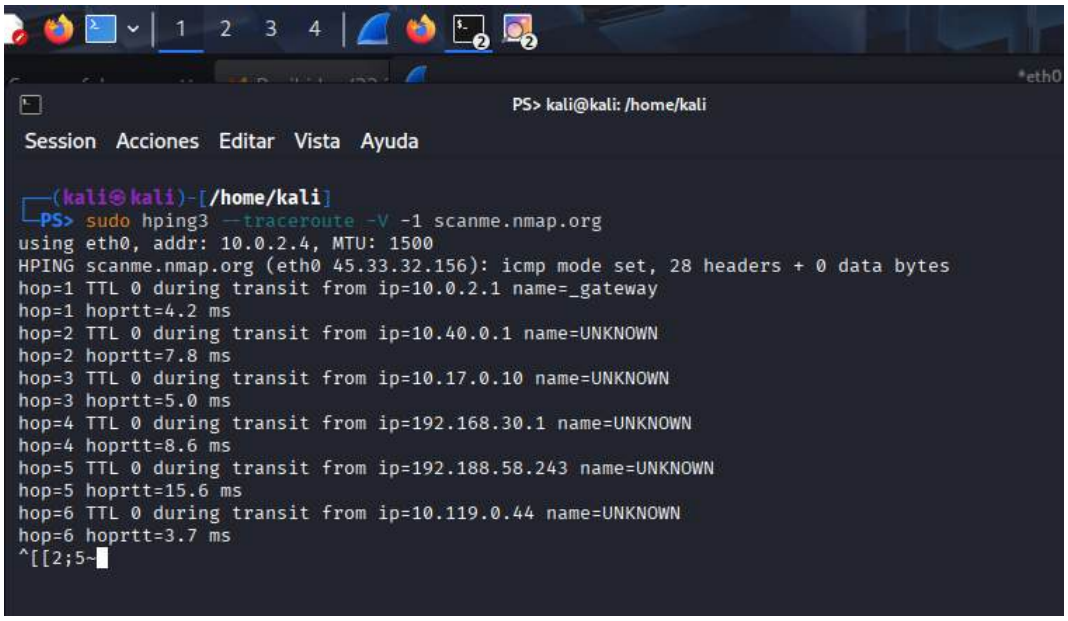
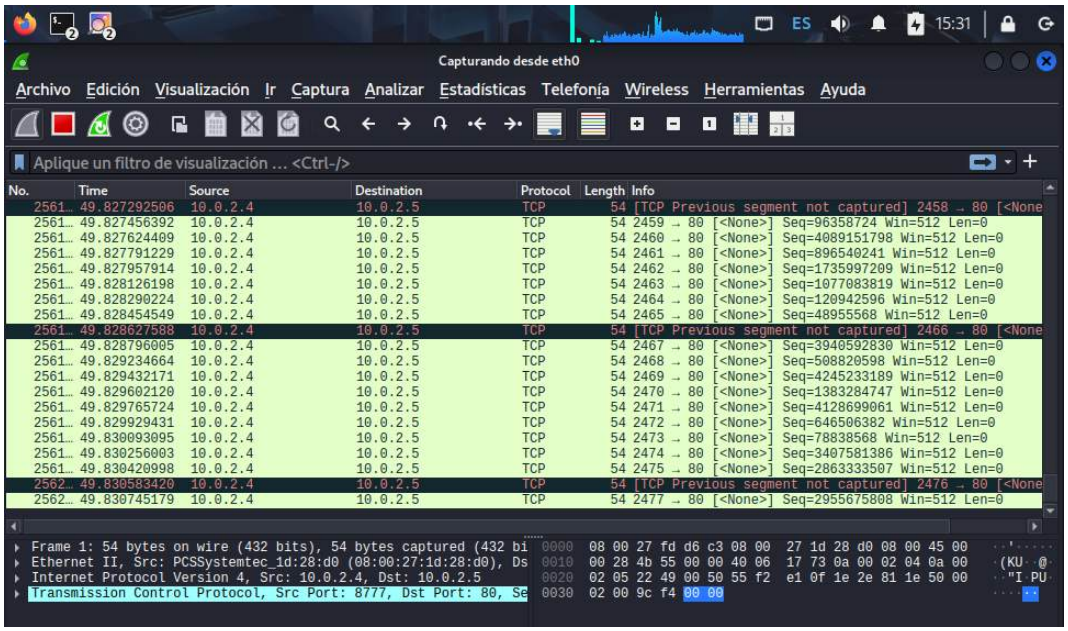


Figura XIII.5: Ataque DoS mediante hping3.



Ejercicios resueltos

En este apartado se presentan 10 ejercicios de ataques de escaneo de puertos con nmap, y de denegación de servicio con hping3.

1. **Ejercicios con nmap**

   a) Realice un escaneo de puertos rápido, en la que se conozca *las versiones de los servicios o aplicaciones*:

```
nmap -sV -F scanme.nmap.org
```

b) Realice un escaneo de puertos *en la que se active el modo verbose (detallado) mediante un SYN sigiloso, y se conozca el sistema operativo de los equipos vecinos*:

```
nmap -v -sS -O scanme.nmap.org/24
```

c) Realice un *escaneo de puertos SSH, DNS, SMTP, IMAP, Aplicaciones específicas, en la que se conozca las versiones*:

```
nmap -sV -p 22  53  110  143  4564 scanme.nmap.org
```

2. **Ejercicios con hping3**

a) Realice una *inundación de paquetes ICMP a la VM Metasploitable 2 dirigido al servidor Web*:

```
hping3 10.0.2.5 –p 80 -"flood
```

b) Realice *una suplantación (spoofing) de la IP real para que sea invisible por medio de una múltiple generación de peticiones de forma que el servidor SSH colapse rápidamente*:

```
hping3 –round-source –flood -p 22 10.0.2.5
```

c) Realice un envío de *solicitud de conexión (SYN) con envío de paquetes más rápido al puerto HTTPS*:

```
hping3 -S –fast 10.0.2.5 -p 443
```

Ejercicios propuestos

Ejercicio 1

Investigue la Captura la Bandera (Capture the flag, CTF) como una competencia de ciberseguridad en la que los participantes resuelven problemas de diversa complejidad, generalmente cadenas de texto ocultas en servicios, archivos o scripts expuestos. Le recomendamos visitar los siguientes enlaces:

picoCTF. (2024). Free Cybersecurity Games. Carnegie Mellon University. `https://picoctf.org/`

CCN-CERT. (2024). Atenea Plataforma de formación. `https://atenea.ccn-cert.cni.es`

Ejercicio 2

El riesgo en Ciberseguridad representa la probabilidad de que una amenaza actúe en contra de una vulnerabilidad. Se le recomienda visitar el siguiente enlace: `www.iso27000,es`

Ejercicio 3

¿Cuáles son los niveles de Ciberseguridad, según la ISO 27032?

Ejercicio 4

Investigue el código César como un mecanismo de criptografía simétrica

Ejercicio 5

Investigue en qué consiste OpenSSL en Linux, como una plataforma de algoritmos criptográficos.

Ejercicio 6

Elabore una matriz que permita determinar con argumentos las diferencias entre criptografía simétrica y asimétrica.

Ejercicio 7

Inyecte paquetes ICMP utilizando la herramienta hping3, desde una máquina en Kali Linux hacia una VM Metasploitable 2.

Ejercicio 8

¿En qué consiste un ataque de escaneo de puertos y revisión de las versiones de las aplicaciones instaladas?.

Ejercicio 9

Investigue diferentes alternativas o modificadores de nmap para realizar el análisis de vulnerabilidades en equipos con sistemas operativos Linux.

Ejercicio 10

Investigue firewall en Linux, y sus semejanzas con el uncomplicated firewall (ufw)

Conclusiones del capítulo

Este capítulo ha proporcionado una visión general de la seguridad de la información y de la ciberseguridad en entornos Linux. La ciberseguridad es la responsable de la seguridad en las redes de computadores, redes celulares, de telecomunicaciones y en el Internet. Los lectores han aprendido en qué consisten los objetivos de la ciberseguridad, algunas definiciones importantes, los mecanismos como los firewalls y la criptografía en forma general. La Ciberseguridad permite a los administradores, técnicos y usuarios incrementar los niveles de seguridad. Por lo tanto, es esencial que aprendan sobre los tipos de ataques, atacantes y mecanismos que pueden ser aplicados a nivel de sistemas operativos. Se han explorado dos tipos de ataques cibernéticos como los de escaneo de puertos mediante nmap y los ataques de denegación de servicio mediante Hping3 utilizando un entono virtual de red Linux controlado. Los ejemplos prácticos y problemas resueltos proporcionados a lo largo del capítulo han permitido consolidar estos conocimientos, ayudando al lector a desarrollar habilidades para simular ataques a redes IP.

Lecciones aprendidas

- No existe un porcentaje del 100 % de seguridad. Lo que corresponde al personal técnico y a los usuarios es minimizar los riesgos controlando las amenazas y las vulnerabilidades de los sistemas operativos.
- La gestión de vulnerabilidades es un proceso crítico y permanente para reducir las amenazas y en consecuencia los riesgos. Este proceso se basa en las primeras fases del Hacking Ético.
- La Criptografía es una disciplina que estudia la forma de cifrar un mensaje para que no sea interceptado y descifrado por un delincuente cibernético (criptoanálisis).
- Las funciones hash son algoritmos matemáticos que transforman una entrada de datos en una cadena de longitud fija, representando la huella digital del contenido.
- Una de las funciones prioritarias de los futuros profesionales dado el incremento de los ataques cibernéticos, es la configuración de seguridad en Sistemas Operativos. Por lo tanto, la formación y entrenamiento en la gestión de la seguridad de la información y de la ciberseguridad en sistemas operativos debe ser permanente y a un buen nivel.

Sitios de interés

Kali Linux como la Distribución de pruebas de penetración más avanzada de Linux. Es de código abierto basada en Debian orientada a diversas tareas de seguridad de la información, como pruebas de penetración, informática forense e ingeniería inversa. Para más información `www.kali.org`

El Código Orgánico Integral Penal (COIP), reglamentos y normas legales que tipifican los delitos informáticos en el Ecuador. Puede ser descargado de: `https://www.oas.org/juridico/PDFs/mesicic5_ecu_ane_con_judi_c%C3%B3d_org_int_pen.pdf`

La ISO 27000 en español: `https://www.iso27000.es/`

Autoevaluación del capítulo

Responda las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿OWASP?**
   - Open Web Application Security Project .
   - Open Worldwide Application Security Project
   - Open-Source Code for Web Applications.
   - Open Web Application Security Project.
2. **Defina Confidencialidad**
   - Define que la última información almacenada no puede ser modificada por usuarios sin permisos.
   - Asegura que tanto la información transmitida como sus claves de acceso, sean secretas y no difundidas.
   - Sostiene que los servicios y datos estén accesibles cuando se necesiten en un 99.999 % por parte de los usuarios.
   - Es la posibilidad de verificar el histórico de las acciones preliminares que se produjeron en un evento, y que constituye evidencia para recuperar un proceso o sistema.
3. **¿Cuál es la función de un firewall en Linux?**
   - Todas las anteriores
   - Permite o bloquea paquetes de datos según reglas predefinidas para proteger el sistema de accesos no autorizados y ataques maliciosos.
   - Es un dispositivo electrónico que tiene esta funcionalidad.
   - Un firewall puede ser software, personas o equipos de hardware.
4. **¿La Criptografía simétrica y asimétrica son diferentes?**
   - La Criptografía asimétrica es más rápida, pues posee menor complejidad algorítmica que la asimétrica.

- La Criptografía simétrica emplea la misma clave privada para encriptar y desencriptar, mientras que la asimétrica dos claves, una pública y otra privada para encriptar desencriptar.
- La criptografía simétrica utiliza la factorización de grandes números primos mientras que la asimétrica utiliza aritmética modular, y operaciones de sustitución y permutación.
- La criptografía asimétrica se basa en funciones hash.

5. **¿En qué se diferencia la identificación de la autenticación?**

   - Las dos significan lo mismo y tienen que ver con el control de acceso de los usuarios.
   - La autenticación es más prioritaria que la autenticación.
   - La identificación le indica al sistema de información quien está accediendo, mientras que la autenticación demuestra que es un usuario legítimo.
   - La identificación y la autenticación forman parte de la triada CID.

6. **¿Qué es un ataque de Denegación de Servicios?**

   - Es un ataque que se realiza intentando descubrir las claves de acceso.
   - Es un ataque mediante el cual el atacante intenta colapsar un servidor mediante la inyección de paquetes infectados, con el fin de impedir que usuarios legítimos puedan acceder a su servicio.
   - Se trata de una técnica que permite ataques a las aplicaciones Web.
   - Se trata de un ataque que se basa en el escaneo de puertos.

7. **El modelo de seguridad de Google:**

   - Significa confianza basada en seguridad perimetral
   - Es un modelo de seguridad confianza cero (zero trust).
   - Es un modelo de seguridad basada en firewall de nueva generación.
   - Es un modelo de seguridad basada en SIEMs.

8. **Defina resiliencia en seguridad de la información**

- Es la propiedad que tiene un sistema de ponerse en funcionamiento en el menor tiempo posible cuando ha sido colapsado mediante un ataque cibernético.
- Es la posibilidad de verificar el histórico de las acciones preliminares que se produjeron en un evento, y que constituye evidencia para recuperar un proceso o sistema.
- Aquella significa que tanto el mensaje como las claves de acceso deben ser secretas.
- Ninguna respuesta es correcta.

9. **¿En qué consiste el análisis y monitoreo de tráfico?**

   - Es una responsabilidad tanto de los administradores de redes, como también de los usuarios para conocer sobre la utilización del procesador, la memoria y HDD.
   - Otros técnicos analizan las posibles amenazas y vulnerabilidades en la red.
   - Es una importante actividad de los administradores de redes o de los usuarios finales para conocer la frecuencia del tráfico en la red, los tipos de protocolos y paquetes, con el fin de controlar una adecuada capacidad de transmisión que garantice el funcionamiento del sistema.
   - Aquella actividad que se realiza para actualizar parches en el sistema operativo.

10. **¿Cuál es la acción del siguiente código: iptables -A OUTPUT -p tcp –dport 80 -j DROP ?**

    - Limita el acceso a páginas web que usan solo el puerto 8080.
    - Limita el acceso a páginas web que usan solo HTTPS.
    - Limita el acceso a páginas web que usan solo el protocolo HTTP.
    - Ninguna de las anteriores.

CAPÍTULO XIV

Microservicios y Contenedores en Linux

Introducción

Las arquitecturas de microservicios han transformado el desarrollo de aplicaciones modernas al permitir la construcción de sistemas compuestos por servicios independientes, escalables y de fácil mantenimiento. Este enfoque favorece la flexibilidad y la rápida evolución de proyectos, especialmente en entornos distribuidos y de computación en la nube [61].

De manera complementaria, la contenerización proporciona mecanismos de aislamiento y portabilidad que facilitan el despliegue consistente de aplicaciones en diferentes plataformas. Tecnologías como Docker y Kubernetes han impulsado la adopción de microservicios al simplificar la gestión, escalabilidad y operación de los servicios [107].

Los sistemas operativos basados en Linux constituyen la plataforma predominante para la implementación de contenedores y servicios distribuidos, debido a que incorporan mecanismos nativos de aislamiento, control de recursos y virtualización ligera. Estas características han convertido a Linux en el entorno de referencia para el desarrollo, despliegue y administración de aplicaciones modernas basadas en microservicios.

El objetivo de este capítulo es presentar los fundamentos de las arquitecturas de microservicios y las tecnologías de contenedores, así como su implementación y administración en sistemas operativos modernos.

En el desarrollo del capítulo usted encontrará tanto el contenido teórico como práctico para su mejor comprensión. Además, se presentan varios ejercicios propuestos para la aplicabilidad práctica.

¡Bienvenidos!

Objetivos de los microservicios y la contenerización

Las arquitecturas de microservicios y las tecnologías de contenerización tienen como finalidad mejorar la construcción, despliegue y mantenimiento de aplicaciones modernas. Su adopción permite desarrollar sistemas más flexibles, escalables y fáciles de administrar, especialmente en entornos Linux y de computación en la nube.

Entre las herramientas más utilizadas para implementar estos conceptos se encuentra Docker, una plataforma que facilita la creación y gestión de contenedores. Gracias a ella es posible ejecutar aplicaciones de manera aislada y portable, aprovechando eficientemente los recursos del sistema operativo.

Virtualización basada en contenedores

Las plataformas de virtualización basadas en contenedores permiten empaquetar una aplicación, junto con todas sus dependencias, como código, bibliotecas y configuraciones, en contenedores [129]. Estos contenedores son entornos ligeros y aislados que comparten el núcleo del sistema operativo del equipo anfitrión. En cambio, las máquinas virtuales utilizan todos los recursos del hardware segmentado lógicamente. En otras palabras, los contenedores son portátiles y rápidos, ya que no requieren un sistema operativo alojado completo, lo que permite que varias aplicaciones se ejecuten de forma consistente en el mismo servidor, lo cual es ideal para microservicios y flujos de trabajo DevOps [128]. A continuación se describen algunas tecnologías, sus características, beneficios y desventajas.

Tecnologías de Contenedores

Docker: Como se mencionó anteriormente, es una plataforma de contenedores ampliamente utilizada, que permite a los desarrolladores empaquetar aplicaciones con todas sus dependencias de contenedores. Los contenedores se pueden implementar en cualquier sistema que admita Docker, lo que garantiza la portabilidad y la coherencia en diferentes entornos [123].

El nombre y logotipo de Docker están representados por una ballena transportando múltiples contenedores, simbolizando la capacidad de ejecutar diversas aplicaciones aisladas sobre un mismo sistema operativo. La Figura XIV.1 muestra la identidad visual de Docker.

Figura XIV.1: Ícono de la Aplicación Docker.

Kubernetes: Es una plataforma estación de orquestación de contenedores que automatiza la implementación, administración y escalabilidad de aplicaciones en contenedores. Aunque no es un sistema de contenedores, es una herramienta crucial para la gestión de contenedores en producción [90].

Contenedores Linux (LXC): Es una tecnología de contenedor que proporciona un entorno virtual para ejecutar múltiples instancias de Linux en un solo núcleo. Es una de las primeras tecnologías de contenedores que utiliza espacios de nombres y grupos de control para el aislamiento de procesos y recursos [112].

Como se muestra en la Figura XIV.2, Docker, Kubernetes y LXC representan algunas de las tecnologías más utilizadas dentro del ecosistema de contenedores, desempeñando funciones complementarias para el despliegue, la administración y la ejecución de aplicaciones modernas.

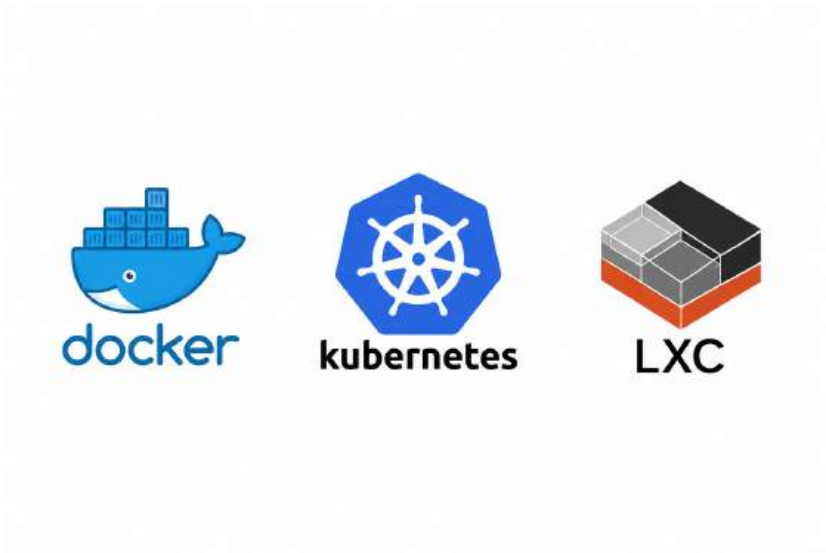


Figura XIV.2: *Principales tecnologías utilizadas en el ecosistema de contenedores: Docker, Kubernetes y LXC.*
Fuente: Esta ilustración fue creada por los autores utilizando ChatGPT.

Características de los contenedores

Los contenedores son artefactos de software que permiten empaquetar y aislar virtualmente aplicaciones para su implementación [39]. A continuación, se detallan sus características clave:

- **Compartir kernel del sistema operativo** Todos los contenedores comparten el mismo kernel del sistema operativo host. No existe un hipervisor entre el hardware y el sistema operativo, lo que reduce la sobrecarga en comparación con la virtualización completa, donde cada máquina virtual tiene su propio sistema operativo alojado.

- **Aislamiento de Procesos y Recursos** Los contenedores y el sistema operativo host están aislados entre sí, a pesar de compartir el mismo núcleo. Esta tecnología garantiza que los procesos en un contenedor no interfieran con los de otro. Utilizan espacios de nombres y grupos de control para administrar y aislar recursos como CPU, memoria, disco, red y dispositivos de entrada y salida para lograrlo.

- **Eficiencia y rendimiento** Debido a la ausencia de un hipervisor y de un núcleo compartido, los contenedores funcionan de manera similar a la virtualización nativa. Los contenedores comienzan y terminan más rápido que las máquinas virtuales completas. Los contenedores son más livianos que las máquinas virtuales porque no incluyen un sistema operativo alojado. Solo contienen las aplicaciones necesarias y sus dependencias. Esta característica permite más contenedores en el mismo hardware en comparación con la cantidad de máquinas virtuales posibles.

- **Escalabilidad:** Los contenedores cuentan con la capacidad de aumentar o disminuir las instancias en varios nodos del clúster, sin afectar su desempeño. Otras tecnologías como Kubernetes también disponen de la capacidad de administrar la escalabilidad y la instanciación de contenedores.

- **Portabilidad:** Los contenedores se pueden empaquetar con todas sus dependencias y configuraciones, lo que los hace portátiles entre diferentes entornos, desde la máquina de desarrollo hasta el servidor de producción. Las imágenes de contenedor permiten la replicación exacta del entorno en diferentes máquinas.

Beneficios

Los contenedores tienen varias ventajas sobre los métodos de virtualización tradicionales. Debido a que son más livianos y portátiles que las máquinas virtuales, los contenedores admiten la descomposición de una aplicación monolítica en microservicios. Los contenedores también se administran e implementan más rápido que las máquinas virtuales, lo que puede ahorrar tiempo y dinero al implementar aplicaciones. Sin embargo, las principales desventajas se enumeran a continuación. Los contenedores son una herramienta poderosa que se puede utilizar para mejorar el desarrollo web y la arquitectura de microservicios.

Desventajas

Seguridad: Aunque los contenedores proporcionan un excelente nivel de aislamiento, todos comparten el mismo núcleo del sistema operativo host. Esto significa que una vulnerabilidad en el kernel puede comprometer potencialmente todos los contenedores del sistema.

Compatibilidad del sistema operativo: Dado que todos los contenedores comparten el mismo núcleo del sistema operativo. Es una plataforma estación de orquestación de con host, solo pueden ejecutar aplicaciones compatibles con el mismo tipo de sistema operativo. Por ejemplo, los contenedores también deben ser Linux en un host Linux. Este aspecto limita la capacidad de ejecutar aplicaciones de diferentes sistemas operativos en el mismo host.

Recursos de hardware: Aunque los contenedores son más livianos que las máquinas virtuales completas, todos los contenedores comparten los mismos recursos de hardware del host, por lo que puede provocar problemas de contención de recursos si muchos contenedores consumen una gran cantidad de CPU, memoria o dispositivos de entrada y salida.

Microservicios en entornos Linux

Los microservicios constituyen un enfoque moderno para el desarrollo de aplicaciones distribuidas, donde una solución de software se divide en múltiples servicios independientes que colaboran entre sí para ofrecer una funcionalidad completa. Este paradigma ha ganado popularidad debido a la necesidad de construir sistemas escalables, flexibles y fáciles de mantener en entornos de computación modernos [71].

Linux se ha consolidado como la plataforma predominante para la implementación de arquitecturas basadas en microservicios gracias a su estabilidad, seguridad, eficiencia en el uso de recursos y amplio soporte para tecnologías de virtualización y contenedores [36, 71].

Concepto de microservicios

La arquitectura de microservicios organiza una aplicación como un conjunto de servicios pequeños e independientes, donde cada uno implementa una funcionalidad específica del negocio. Estos servicios se comunican mediante protocolos de red, generalmente utilizando interfaces de programación de aplicaciones (API) basadas en HTTP o protocolos de mensajería.

A diferencia de las aplicaciones monolíticas, donde todos los componentes se encuentran integrados en una única unidad de ejecución, los microservicios permiten desarrollar, desplegar y actualizar cada componente de manera independiente [9]. Como se muestra en la Figura XIV.3, la arquitectura monolítica concentra toda la lógica de la aplicación en una única unidad, mientras que la arquitectura de microservicios divide la funcionalidad en servicios independientes que pueden desplegarse y escalarse de forma individual.

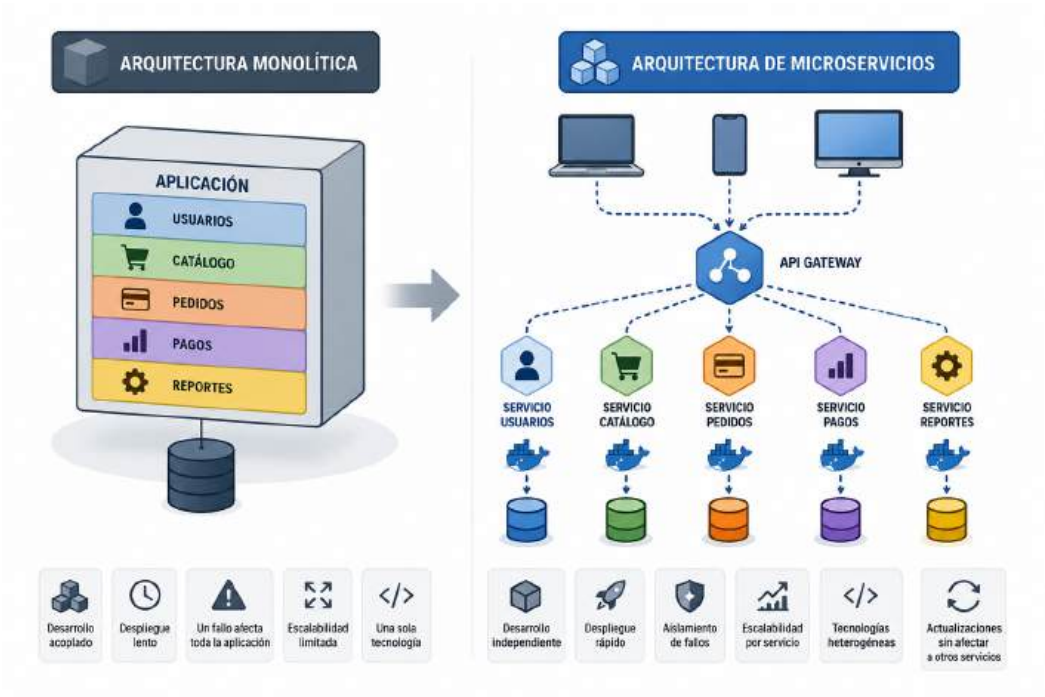


Figura XIV.3: *Comparación entre una arquitectura monolítica y una arquitectura basada en microservicios.*
Fuente: Esta ilustración fue creada por los autores utilizando ChatGPT.

Linux como plataforma para microservicios

Linux proporciona las capacidades necesarias para ejecutar aplicaciones distribuidas de forma eficiente. Entre sus principales características destacan la gestión avanzada de procesos, el soporte para redes, la administración de memoria y los mecanismos de seguridad integrados.

Además, la mayoría de las plataformas de nube pública y centros de datos utilizan distribuciones Linux como sistema operativo base, lo que ha favorecido su adopción en arquitecturas modernas basadas en microservicios [71].

Contenedores y microservicios

Los contenedores representan una de las tecnologías más importantes para la implementación de microservicios. Un contenedor encapsula una aplicación junto con sus dependencias, bibliotecas y configuraciones necesarias para su ejecución.

Docker es una de las plataformas más utilizadas para la creación y administración de contenedores. Gracias a esta tecnología, los desarrolladores pueden garantizar que una aplicación funcione de manera consistente en diferentes entornos de ejecución [88, 36]. La Figura XIV.4 presenta una representación conceptual de la relación entre Linux, los contenedores y los microservicios.

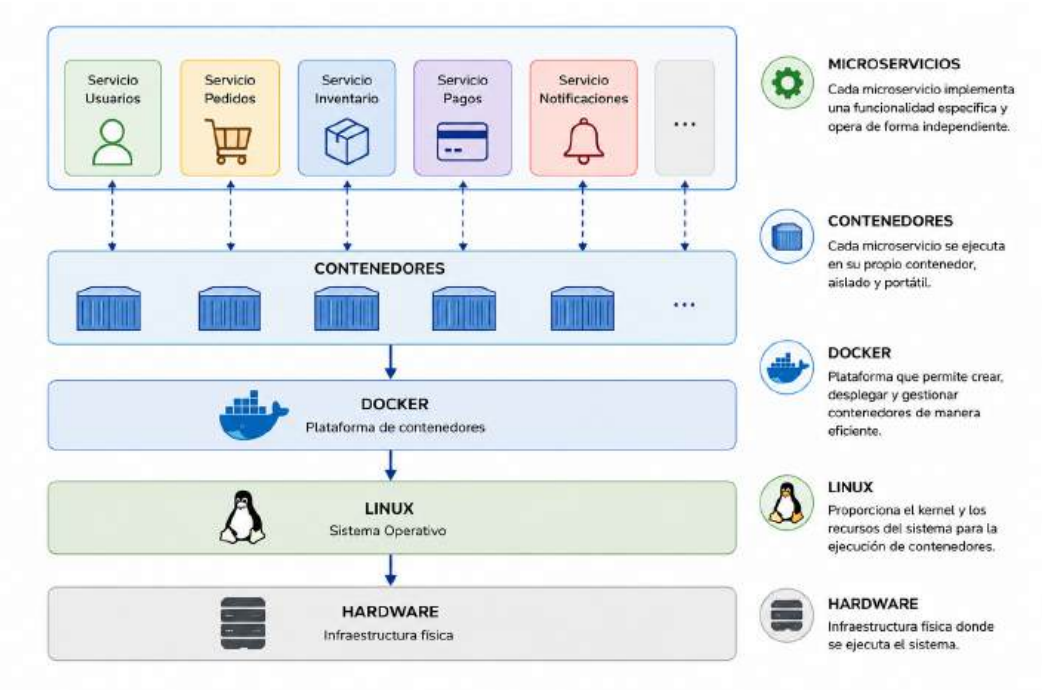


Figura XIV.4: *Relación entre Linux, los contenedores y los microservicios.*
Fuente: Esta ilustración fue creada por los autores utilizando ChatGPT.

Orquestación mediante Kubernetes

A medida que aumenta el número de microservicios desplegados, la administración manual de contenedores se vuelve compleja. Para resolver este problema surgen plataformas de orquestación como Kubernetes.

Kubernetes permite automatizar tareas como el despliegue de aplicaciones, el balanceo de carga, la recuperación ante fallos y el escalamiento automático de servicios. Estas capacidades facilitan la gestión de aplicaciones distribuidas ejecutadas sobre Linux [136].

Ventajas de los microservicios

La adopción de arquitecturas basadas en microservicios ofrece múltiples beneficios:

- Escalabilidad independiente de cada servicio.
- Mayor flexibilidad para el desarrollo y mantenimiento.
- Reducción del impacto de fallos.
- Actualización continua de componentes específicos.
- Compatibilidad con prácticas DevOps y CI/CD.

Estas ventajas han impulsado su adopción en aplicaciones empresariales, plataformas de comercio electrónico, sistemas financieros y servicios en la nube [26].

Desafíos de los microservicios

A pesar de sus beneficios, los microservicios también presentan desafíos técnicos relacionados con la administración de múltiples servicios, la observabilidad, la seguridad de las comunicaciones y la gestión distribuida de datos.

Por esta razón, la implementación de arquitecturas de microservicios suele complementarse con herramientas de monitoreo, automatización y orquestación especializadas [26, 9].

Ejercicios resueltos

En este apartado se presentan 10 ejercicios relacionados con la administración de contenedores mediante Docker y la orquestación de aplicaciones utilizando Kubernetes.

1. **Ejercicios con Docker**

   a) Verifique que Docker se encuentra instalado correctamente y consulte su versión.

   Verificar la instalación de Docker

   ```
   docker -version
   ```

   b) Descargue la imagen oficial de Ubuntu desde Docker Hub.

   Descargar una imagen desde Docker Hub

   ```
   docker pull ubuntu
   ```

   c) Liste todas las imágenes almacenadas localmente.

   Visualizar imágenes disponibles

   ```
   docker images
   ```

   d) Cree y ejecute un contenedor interactivo basado en Ubuntu.

   Ejecutar un contenedor interactivo

   ```
   docker run -it ubuntu bash
   ```

   e) Visualice todos los contenedores en ejecución.

   Listar contenedores en ejecución

   ```
   docker ps
   ```

2. **Ejercicios con Kubernetes**

   a) Verifique que el clúster de Kubernetes se encuentre disponible.

   Consultar el estado del clúster

   ```
   kubectl cluster-info
   ```

   b) Liste todos los nodos del clúster.

   Listar nodos del clúster

   ```
   kubectl get nodes
   ```

Ejercicios propuestos

Ejercicio 1

Explique las principales diferencias entre una máquina virtual y un contenedor considerando el consumo de recursos, el tiempo de inicio y la portabilidad.

Ejercicio 2

Investigue las funciones principales de Docker Hub y explique cómo facilita la distribución de imágenes de contenedores.

Ejercicio 3

Investigue cuáles son las diferencias entre Docker Compose y Kubernetes, indicando en qué escenarios es recomendable utilizar cada uno.

Ejercicio 4

Desarrolle un Dockerfile para una aplicación sencilla (por ejemplo, una aplicación en Python o Node.js), construya la imagen y ejecútela en un contenedor.

Ejercicio 5

Investigue los mecanismos de aislamiento utilizados por Linux en la ejecución de contenedores, incluyendo los conceptos de Namespaces y Control Groups (cgroups).

Conclusiones del capítulo y lecciones aprendidas

Este capítulo ha presentado los fundamentos de las arquitecturas de microservicios y las tecnologías de contenerización empleadas en el desarrollo moderno de aplicaciones. Los microservicios permiten dividir sistemas complejos en componentes independientes, facilitando su mantenimiento, escalabilidad y evolución.

Asimismo, se estudió el papel de Linux como plataforma base para la ejecución de contenedores, aprovechando mecanismos nativos de aislamiento, administración de recursos y virtualización ligera. Herramientas como Docker

simplifican el empaquetado y despliegue de aplicaciones, garantizando consistencia entre los entornos de desarrollo, pruebas y producción.

Los ejercicios prácticos desarrollados permitieron al lector familiarizarse con la creación, gestión y ejecución de contenedores, así como comprender las ventajas que ofrece la contenerización frente a otros mecanismos tradicionales de virtualización.

Al finalizar este capítulo, el lector comprenderá los principios fundamentales de los microservicios y la contenerización, además de poseer los conocimientos básicos necesarios para implementar y administrar aplicaciones contenerizadas en sistemas Linux.

Lecciones aprendidas

- Los microservicios permiten construir aplicaciones mediante servicios independientes que pueden desarrollarse, desplegarse y mantenerse de forma autónoma.
- Los contenedores proporcionan entornos aislados que incluyen las dependencias necesarias para ejecutar una aplicación de manera consistente.
- Linux constituye la plataforma predominante para la implementación de contenedores debido a sus mecanismos nativos de aislamiento y gestión eficiente de recursos.
- Docker facilita la creación, distribución y ejecución de contenedores, simplificando el despliegue de aplicaciones en diferentes entornos.
- Los contenedores consumen menos recursos que las máquinas virtuales tradicionales al compartir el núcleo del sistema operativo anfitrión.
- La contenerización mejora la portabilidad, escalabilidad y disponibilidad de las aplicaciones modernas.

Sitios de interés

Sitio web oficial de Docker, `https://www.docker.com`, donde se encuentra documentación, tutoriales y herramientas para la creación y administración de contenedores.

Sitio web oficial de Docker Documentation, `https://docs.docker.com`, que contiene guías técnicas, referencias de comandos y ejemplos prácticos para el despliegue de aplicaciones contenerizadas.

Sitio web oficial de Kubernetes, `https://kubernetes.io`, proyecto de código abierto orientado a la automatización del despliegue, escalado y administración de aplicaciones basadas en contenedores.

Sitio web oficial de Podman, `https://podman.io`, herramienta para la gestión de contenedores compatible con los estándares OCI y ampliamente utilizada en entornos Linux.

Sitio web oficial de The Linux Foundation, `https://www.linuxfoundation.org`, organización encargada de promover el desarrollo de Linux y múltiples proyectos de software libre relacionados con infraestructura y computación en la nube.

Autoevaluación del capítulo

Responda las siguientes preguntas para evaluar la comprensión de los conceptos abordados en este capítulo. Las respuestas correctas están resaltadas en color azul.

1. **¿Qué es un contenedor?**
   - Es una máquina virtual completa con su propio sistema operativo.
   - Es un servidor físico dedicado.
   - Es un entorno aislado que empaqueta una aplicación junto con todas sus dependencias.
   - Es una partición del disco duro.
2. **¿Cuál es una ventaja de los contenedores frente a las máquinas virtuales?**
   - Consumen más memoria RAM.
   - Requieren un sistema operativo independiente para cada instancia.

- Comparten el kernel del sistema operativo y utilizan menos recursos.
- Solo pueden ejecutarse en Windows.

3. **¿Cuál es la función principal de Docker?**
   - Administrar bases de datos.
   - Crear máquinas virtuales.
   - Crear, distribuir y ejecutar aplicaciones dentro de contenedores.
   - Administrar usuarios del sistema operativo.

4. **¿Cuál es la función principal de Kubernetes?**
   - Crear imágenes Docker.
   - Ejecutar aplicaciones de escritorio.
   - Automatizar el despliegue, la administración y el escalamiento de contenedores.
   - Reemplazar el sistema operativo Linux.

5. **¿Qué caracteriza a una arquitectura de microservicios?**
   - Toda la aplicación se ejecuta como un único bloque.
   - La aplicación está dividida en servicios pequeños e independientes.
   - Solo puede utilizar bases de datos distribuidas.
   - Requiere una máquina virtual para cada servicio.

6. **¿Cuál es una diferencia entre una arquitectura monolítica y una arquitectura de microservicios?**
   - En una arquitectura monolítica cada servicio se despliega por separado.
   - En una arquitectura monolítica todos los componentes forman una sola aplicación, mientras que en microservicios se implementan de manera independiente.
   - No existe ninguna diferencia entre ambas arquitecturas.
   - Los microservicios solo funcionan en Windows.

7. **¿Por qué Linux es ampliamente utilizado para implementar microservicios?**

   - Porque únicamente funciona con Docker.
   - Porque consume más recursos que otros sistemas operativos.
   - Porque solo admite aplicaciones web.
   - Porque ofrece estabilidad, seguridad, eficiencia y soporte para tecnologías de contenedores.

8. **¿Cuál de las siguientes tecnologías corresponde a una plataforma de contenedores?**

   - Apache.
   - PostgreSQL.
   - Docker.
   - Git.

9. **¿Cuál es una de las principales ventajas de utilizar microservicios?**

   - Todos los servicios deben actualizarse simultáneamente.
   - Solo permiten aplicaciones pequeñas.
   - Cada servicio puede desarrollarse, desplegarse y escalarse de manera independiente.
   - Obligan a utilizar una única base de datos.

10. **¿Cuál es una de las principales desventajas de los contenedores?**

    - No permiten ejecutar aplicaciones.
    - Siempre requieren un hipervisor.
    - Comparten el kernel del sistema operativo, por lo que una vulnerabilidad en este puede afectar a varios contenedores.
    - No pueden utilizar redes.

Referencias

[1] Marcelo Abranches, Erika Hunhoff, Rohan Eswara, Oliver Michel, and Eric Keller. Linuxfp: Transparently accelerating linux networking. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, page 543–554. IEEE, July 2024.

[2] M. Anandha Meenachi, Deepa Karuppaiah, and Saravanakumar Kandasamy. *A Comparative Analysis of Prominent Operating Systems: Windows, Android, Linux, and Others*, page 581–593. Springer Nature Singapore, November 2025.

[3] Howard Anderson and Mike Tooley. *Newnes PC troubleshooting pocket book*. Elsevier, 2003.

[4] Apple. Remembering steve jobs. `https://www.apple.com/stevejobs/`, 2011.

[5] Apple. Use filevault to encrypt the startup disk on your mac. `https://support.apple.com/HT204837`, 2025.

[6] Oluwasanmi Richard Arogundade and Dr. Kiran Palla. Virtualization revolution: Transforming cloud computing with scalability and agility. *IARJSET*, 10(6), June 2023.

[7] Jacek Artymiak. *Beginning OpenOffice Calc: From Setting Up Simple Spreadsheets to Business Forecasting*. Apress, 2011.

[8] Syed Khundmir Azmi. Cryptographic hashing beyond sha: Designing collision-resistant, quantum-resilient hash functions. *International Journal of Science and Research Archive*, 12(2):3119–3127, 2024.

[9] Luciano Baresi, Giovanni Quattrocchi, and Damian Andrew Tamburri. Microservice architecture practices and experience: a focused look on docker configuration files. *arXiv preprint arXiv:2212.03107*, 2022.

[10] Komal Barge and Vijaya Kamble. A prediction of operating systems vulnerabilities using machine learning algorithms. In *FIFTH INTERNATIONAL CONFERENCE ON APPLIED SCIENCES: ICAS2023*, volume 3097, page 020247. AIP Publishing, 2024.

[11] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177. ACM, 2003.

[12] Muli Ben-Yehuda, Eric Van Hensbergen, and Marc Fiuczynski. Minding the gap: Ramp;d in the linux kernel. *ACM SIGOPS Operating Systems Review*, 42(5):1–3, July 2008.

[13] Richard Blum and Christine Bresnahan, 2015.

[14] Encyclopaedia Britannica. Bill gates - biography, microsoft cofounder, & philanthropist. `https://www.britannica.com/money/Bill-Gates`.

[15] Encyclopaedia Britannica. Linus torvalds - biography, linux, & facts. `https://www.britannica.com/biography/Linus-Torvalds`.

[16] David Brown and Emily White. A comparative study of scheduling algorithms for high-performance systems. *High-Performance Computing Journal*, 24(3):78–95, 2014.

[17] Ioana BRĂNESCU and Emil SIMION. Mac-os ransomware protection: Prevention and detection mechanisms. *Romanian Cyber Security Journal*, 5(1):87–95, May 2023.

[18] Rajkumar Buyya, James Broberg, and Andrzej Goscinski. *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, Hoboken, NJ, 2013.

[19] Alexandru Calcatinge and Julian Balog. *Mastering Linux Administration: Take your sysadmin skills to the next level by configuring and maintaining Linux systems*. Packt Publishing Ltd, 2024.

[20] Oswald Campesato. *Bash for Data Scientists*. Walter de Gruyter GmbH & Co KG, 2022.

[21] Jason Cannon. *High Availability for the LAMP Stack: Eliminate Single Points of Failure and Increase Uptime for Your Linux, Apache, MySQL, and PHP Based Web Applications*. CreateSpace Independent Publishing Platform, 2014.

[22] Daniel Carson. The ubuntu operating system. 2020.

[23] Manashree Chakraborty, Kanu Chakraborty, and Navin Upadhyay. A study of the GNU general public licence-based patients information management system to manage healthcare systems. *International Journal of Information Studies*, 15(2):55–58, April 2023.

[24] Pranabananda Chakraborty. *Computer Organisation and Architecture: Evolutionary Concepts, Principles, and Designs*. Chapman and Hall/CRC, 2020.

[25] Pranabananda Chakraborty. *Operating Systems: Evolutionary Concepts and Modern Design Principles*. Chapman and Hall/CRC, 2023.

[26] Ramaswamy Chandramouli et al. Security strategies for microservices-based application systems. Technical Report SP 800-204, National Institute of Standards and Technology (NIST), 2019.

[27] G. Chitralekha. *A Survey on Different Scheduling Algorithms in Operating System*, page 637–651. Springer Nature Singapore, 2021.

[28] Brendan Choi. *Creating an Ubuntu Server Virtual Machine (VM)*, page 271–309. Apress, 2024.

[29] Lluís Codina and Cristòfol Rovira. lt;igt;openofficelt;/igt; y el formato lt;igt;opendocumentlt;/igt;: funciones y compatibilidad. *El Profesional de la Informacion*, 17(4):453–462, 2008.

[30] Raditya Arya Darmawan and Weni Gurita Aedi. Important role of operating system in computer network connection. *Dinasti International Journal of Education Management & Social Science*, 6(2), 2024.

[31] Harvey M. Deitel and Paul J. Deitel. *Operating Systems*. Prentice Hall, 3rd edition, 2004.

[32] Ümit Demirbaga, Gagangeet Singh Aujla, Anish Jindal, and Oğuzhan Kalyon. *Cloud Computing for Big Data Analytics*, page 43–77. Springer Nature Switzerland, 2024.

[33] Chetan Dhule and Urmila Shrawankar. *Impact Analysis of Hypervisors on the Performance of Virtualized Resources*, page 421–427. Springer Singapore, 2021.

[34] Jose Dieguez Castro. *Arch Linux*, page 235–252. Apress, 2016.

[35] Jose Dieguez Castro. *Fedora*, page 73–103. Apress, 2016.

[36] Docker Inc. What is docker?, 2026.

[37] Ubuntu Documentation. The linux command line for beginners. `https://ubuntu.com/tutorials/command-line-for-beginners`.

[38] Elias Dritsas, Maria Trigka, Christos Troussas, and Phivos Mylonas. Multimodal interaction, interfaces, and communication: A survey. *Multimodal Technologies and Interaction*, 9(1):6, January 2025.

[39] Bimal Raj Dutta and Basit Mohi ud din Naikoo. *Design and Analysis of Advance DGS Pi-Slotted Microstrip Patch Antenna for 5G Applications*, page 15–29. Springer Nature Switzerland, August 2025.

[40] Biography.com Editors. Bill gates biography. `https://www.biography.com/business-leaders/bill-gates`.

[41] Irv Englander and Wilson Wong. *The architecture of computer hardware, systems software, and networking: An information technology approach*. John Wiley & Sons, 2021.

[42] Aymé Escobar Díaz, Ricardo Rivadeneira, Richard Albán, and Walter Fuertes. *Hardening Linux Firewall Rules to Detect and Mitigate Vulnerabilities in Node.js Services*, page 267–279. Springer Nature Singapore, 2025.

[43] Douglas Everson and Long Cheng. A survey on network attack surface mapping. *Digital Threats: Research and Practice*, 5(2):1–25, June 2024.

[44] Arturo Fernández. *Cámbiate a LINUX*. RC Libros, 2011.

[45] Linux Foundation. It's not just the linux operating system. `https://www.linuxfoundation.org/blog/blog/the-linux-foundation-its-not-just-the-linux-operating-system`.

[46] MacArthur Foundation. Richard m. stallman – class of 1990. `https://www.macfound.org/fellows/class-of-1990/richard-m-stallman`.

[47] Tulsi Pawan Fowdur and Lavesh Babooram. *Network Traffic Monitoring and Analysis*, page 51–96. Apress, 2024.

[48] Richard Fox. *Linux with operating system concepts*. Chapman and Hall/CRC, 2021.

[49] W. Fuertes, J. E. Lopez de Vergara, F. Meneses, and F. Galan. A generic model for the management of virtual network environments. In *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, page 813–816. IEEE, April 2010.

[50] Walter Fuertes and Jorge López De Vergara M. An emulation of vod services using virtual network environments. *Electronic Communications of the EASST*, 17, 2009.

[51] Walter Fuertes and Mayra Macas. *Ciberseguridad: Del ciber-crimen a los ataques ciber-físicos*. Comisión Editorial ESPE, Sangolquí, Ecuador, 1ra edition, 2023.

[52] Greg Gagne, Abraham Silberschatz, and Peter B. Galvin. *Understanding Operating Systems*. Wiley, 10th edition, 2018.

[53] Evan Gardner, Gurmeet Singh, and Weihao Qu. Penetration testing operating systems: Exploiting vulnerabilities. In *2024 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, page 1–9. IEEE, October 2024.

[54] Riya Gohil and Hiren Patel. Comparative analysis of cloud platform: Amazon web service, microsoft azure, and google cloud provider: A review. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, page 1–5. IEEE, June 2024.

[55] Herschel Gonzales. herschelgonzalez.com. `https://herschelgonzalez.com/que-es-un-sistema-operativo-comercial/`, agosto 8 2021.

[56] Adrián Guayasamín, Walter Fuertes, Nahir Carrera, Luis Tello-Oquendo, and Valeria Suango. Blockchain-empowered e-ticket distribution system for secure and efficient transactions, validation, and audits. *Annals of Telecommunications*, October 2025.

[57] Ruchika Gupta, Pankaj Gupta, and Jaswinder Singh. *Security and Cryptography*, page 501–547. Elsevier, 2019.

[58] Kurt H. Hansen and Fergus Toolan. Decoding the apfs file system. *Digital Investigation*, 22:107–132, September 2017.

[59] Per Brinch Hansen. *The Evolution of Operating Systems*, page 1–34. Springer New York, 2001.

[60] Darren James Harkness. *Apache essentials*. Springer, 2022.

[61] Sara Hassan, Rami Bahsoon, and Rajkumar Buyya. Systematic scalability analysis for microservices granularity adaptation design decisions. *Software: Practice and Experience*, 52(6):1378–1401, 2022.

[62] Tarek Helmy. *Optimizing Round-Robin Scheduling Algorithm Performance in Real-Time Systems*, page 371–382. Springer Nature Switzerland, 2024.

[63] Wim H. Hesselink, Peter A. Buhr, and Colby A. Parsons. First-come-first-served as a separate principle. *ACM Transactions on Parallel Computing*, 11(4):1–20, November 2024.

[64] Chris Hughes and Nikki Robinson. *Effective vulnerability management: managing risk in the vulnerable digital ecosystem*. John Wiley & Sons, 2024.

[65] Manuel José Fernández Iglesias. Conceptos básicos de sistemas operativos. *Universidad de Vigo*, 2023.

[66] Trent Jaeger. *Operating system security*. Springer Nature, 2022.

[67] Ashish Jaiswal. Journal of advances in shell programming. 2022.

[68] Umapathi Janne. *Web Design and CSS Animation*. Blue Rose Publishers, 2024.

[69] Abderrahim Kaabouch, Mohammed Bouhassane, Abdelalim Sadiq, Oussama Aziz, and Mohamed Oualla. A comparative analysis of apache cloud projects for data storage. In *2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet)*, page 1–9. IEEE, December 2024.

[70] Brijender Kahanwal, Tejinder Pal Singh, Ruchira Bhargava, and Girish Pal Singh. File system - a component of operating system, 2013.

[71] Arnon Karmel. Nist definition of microservices, application containers and system virtual machines. Technical Report SP 800-180, National Institute of Standards and Technology (NIST), 2016.

[72] Brian W Kernighan and Rob Pike. The unix programming environment. prentice hall. *Englewood Cliffs, NY*, 1984.

[73] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 1978.

[74] Brian W Kernighan and Dennis M Ritchie. *El lenguaje de programación C*. Pearson Educación, 1991.

[75] Angelos D. Keromytis, Jason L. Wright, Theo De Raadt, and Matthew Burnside. Cryptography as an operating system service: A case study. *ACM Transactions on Computer Systems*, 24(1):1–38, February 2006.

[76] Michael Kerrisk. *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.

[77] Priya Kumari and Nitin Jain. *Performance Enhancement and Scheduling in Communication Networks—A Review into Various Approaches*, page 661–672. Springer Nature Singapore, 2024.

[78] Bell Labs. The evolution of the unix time-sharing system. `https://www.nokia.com/bell-labs/about/dennis-m-ritchie/hist.html`.

[79] Jim Ledin. *Modern computer architecture and organization*, volume 2. Packt Publishing Birmingham, UK, 2020.

[80] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. Scan me if you can: Understanding and detecting unwanted vulnerability scanning. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2284–2294. ACM, April 2023.

[81] Xiaodong Lin. *Examining FAT File System*, page 115–144. Springer International Publishing, 2018.

[82] Mayra Macas, Chunming Wu, and Walter Fuertes. Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems. *Expert Systems with Applications*, 238:122223, March 2024.

[83] Mika Mannila. Free and open source software: Approaches in brazil and argentina. Technical Report 5, Hyper Medialab, Manila, 2005.

[84] A David Garza Marín. *El libro negro de las computadoras en la productividad: Cómo determinar la PC de escritorio o portátil adecuada para sus necesidades productivas*. A. David Garza Marín, 2020.

[85] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report Special Publication 800-145, National Institute of Standards and Technology, 2011.

[86] K. Meneses et al. Evaluation of rsa and ecc for securing embedded systems. *IEEE Latin America Transactions*, 14(5):2452–2457, 2016.

[87] Ric Messier and Michael Jang. *Security strategies in Linux platforms and applications*. Jones & Bartlett Learning, 2022.

[88] Microsoft. What is docker?, 2022.

[89] Manvi Mishra, Md Shadab Hussain, Nirbhay Kumar Chaubey, and Prabhakar Gupta. The role of wireshark in packet inspection and password sniffing for network security. In *Advanced Cyber Security Techniques for Data, Blockchain, IoT, and Network Protection*, pages 225–244. IGI Global Scientific Publishing, 2025.

[90] Marek Moravcik, Martin Kontsek, Pavel Segec, and David Cymbalak. Kubernetes - evolution of virtualization. In *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, page 454–459. IEEE, October 2022.

[91] Christine Murad and Cosmin Munteanu. Designing voice interfaces: Back to the (curriculum) basics. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–12. ACM, April 2020.

[92] Ivan Nedyalkov and Georgi Georgiev. Kali linux – a simple and effective way to study the level of cyber security and penetration testing of power electronic devices. *International Journal on Information Technologies and Security*, 16(2):103–114, 2024.

[93] A Nepal. Linux server & hardening security, 2014.

[94] Xudong Ni, Zhimin Yang, Xiaole Bai, Adam C. Champion, and Dong Xuan. Diffuser: Differentiated user access control on smartphones. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, page 1012–1017. IEEE, October 2009.

[95] Gerard O'Regan. *Overview of Operating Systems*, page 203–215. Springer International Publishing, 2018.

[96] Partho Protim Ghosh. Operating system comparison for fault tolerance, 2020.

[97] Santiago Paz. Cybersecurity standards and frameworks, September 2023.

[98] Y. Judith Petrizia, R. Sujitha, and Sree Saradha M. Overview of process management in operating systems. *International Journal of Advanced Research in Science, Communication and Technology*, page 99–106, February 2025.

[99] Prakhar Pipersania. Scheduling-algorithms. `https://github.com/prakhar-pipersania/Scheduling-Algorithms`, 2020.

[100] Ubuntu Documentation Project. Using the terminal – ubuntu community help wiki. `https://help.ubuntu.com/community/UsingTheTerminal`.

[101] Donagani Ramakrishna and Mohammed Ali Shaik. A comprehensive analysis of cryptographic algorithms: Evaluating security, efficiency, and future challenges. *IEEE Access*, 13:11576–11593, 2025.

[102] Flavio Ramalho and Augusto Neto. Virtualization at the network edge: A performance comparison. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, page 1–6. IEEE, June 2016.

[103] A. U. Rehman, Rui L. Aguiar, and Joao Paulo Barraca. Network functions virtualization: The long road to commercial deployments. *IEEE Access*, 7:60439–60464, 2019.

[104] Dirk Riehle. Free and open source software. *Computer*, 57(8):114–118, August 2024.

[105] Andrey Rodrigues, Maria Lúcia Bento Villela, and Eduardo Luzeiro Feitosa. Privacy threat modeling language. *IEEE Access*, 11:24448–24471, 2023.

[106] Mario Ron, Walter Fuertes, Marco Bonilla, Theofilos Toulkeridis, and Javier Diaz. Cybercrime in ecuador, an exploration, which allows to define national cybersecurity policies. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, page 1–7. IEEE, June 2018.

[107] Abdul Saboor, Mohd Fadzil Hassan, Rehan Akbar, Syed Nasir Mehmood Shah, Farrukh Hassan, Saeed Ahmed Magsi, and Muhammad Aadil Siddiqui. Containerized microservices orchestration and provisioning in cloud computing: A conceptual framework and future perspectives. *Applied Sciences*, 12(12):5793, 2022.

[108] Alireza Sadeghi-Nasab and Vahid Rafe. A comprehensive review of the security flaws of hashing algorithms. *Journal of Computer Virology and Hacking Techniques*, 19(2):287–302, October 2022.

[109] Édgar Salguero Dorokhin, Walter Fuertes, and Edison Lascano. On the development of an optimal structure of tree parity machine for the establishment of a cryptographic key. *Security and Communication Networks*, 2019:1–10, March 2019.

[110] Ameer Sameer. Common linux ubuntu commands overview, 2015.

[111] Olaf Saßnick, Thomas Rosenstatter, Christian Schäfer, and Stefan Huber. Stride-based methodologies for threat modeling of industrial control systems: A review. In *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*, page 1–8. IEEE, May 2024.

[112] S Senthil Kumaran. *Practical LXC and LXD: linux containers for virtualization and orchestration*. Springer, 2017.

[113] Aleatha Shanley and Michael Johnstone. Selection of penetration testing methodologies: A comparison and evaluation. *13th Australian Information Security Management Conference*, held from the 30 November – 2 December:Western Australia., 2015.

[114] Prem Sagar Sharma, Sanjeev Kumar, Madhu Sharma Gaur, and Vinod Jain. A novel intelligent round robin cpu scheduling algorithm. *International Journal of Information Technology*, 14(3):1475–1482, March 2021.

[115] Ravikant Shukla. Why mac-os is safer better than windows. *International Journal for Research in Applied Science and Engineering Technology*, 10(7):777–780, July 2022.

[116] Abraham Silberschatz, Peter Galvin, Greg Gagne, et al. *Sistemas operativos*. México: Editorial Limusa,, 1999.

[117] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, 10th edition, 2018.

[118] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, Hoboken, NJ, 10th edition, 2020.

[119] Siva Raja Sindiramutty, Krishna Raj V. Prabagaran, Rehan Akbar, Manzoor Hussain, and Nazir Ahmed Malik. *Generative AI for Secure User Interface (UI) Design*, page 333–394. IGI Global, September 2024.

[120] Glen D Singh. *The Ultimate Kali Linux Book: Harness Nmap, Metasploit, Aircrack-ng, and Empire for Cutting-edge Pentesting*. Packt Publishing Ltd, 2024.

[121] James E. Smith and Ravi Nair. The architecture of virtual machines. *IEEE Computer*, 38(5):32–38, 2005.

[122] Mark G Sobell. *A practical guide to Linux commands, editors, and shell programming*. Prentice Hall, 2013.

[123] Maciej Sobieraj and Daniel Kotyński. Docker performance evaluation across operating systems. *Applied Sciences*, 14(15):6672, July 2024.

[124] William Stallings. *Operating Systems: Internals and Design Principles*. Pearson, 9th edition, 2018.

[125] Richard Stallman. *Free software, free society: Selected essays of Richard M. Stallman*. Lulu. com, 2002.

[126] Richard M Stallman. Gnu compiler collection internals. *Free Software Foundation*, 46, 2002.

[127] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. Pearson Education, Boston, 4th edition, 2015.

[128] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. From monolithic systems to microservices: A comparative study of performance. *Applied Sciences*, 10(17):5797, August 2020.

[129] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Jorge Edison Lascano, and Theofilos Toulkeridis. A container orchestration development that optimizes the etherpad collaborative editing tool through a novel management system. *Electronics*, 9(5):828, May 2020.

[130] Akhilesh S. Thyagaturu, Prateek Shantharama, Ahmed Nasrallah, and Martin Reisslein. Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies. *IEEE Access*, 10:79825–79873, 2022.

[131] Ravikant Tiwari and Dr-Shadab Siddiqui. Analytical survey of windows operating system and comparison of windows, linux and android operating system. *International Journal of Engineering Applied Sciences and Technology*, 6, 06 2021.

[132] Konstantin Tsvetkov. Operating systems: Past, present and future. *New knowledge Journal of science*, 9(2):167–175, 2020.

[133] John O Ugah, Sunday C. Agu, and Felix Elugwu. Relationship between operating system, computer hardware, application software and other software. *International Journal of Computer Trends and Technology*, 64(1):12–16, October 2018.

[134] Ramiz Valiyev. NTFS file system, 2023.

[135] Sebastiano Vasi, Ulderico Wanderlingh, and Giuseppe Mandaglio. *Operative Systems and Linux Shells*, page 3–18. Springer Nature Switzerland, 2026.

[136] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Kubernetes as an availability manager for microservice applications. *arXiv preprint arXiv:1901.04946*, 2019.

[137] Mark Jeriel Villegas. Centos. *SSRN Electronic Journal*, 2024.

[138] Sheridan Yuen. *Mastering Windows Presentation Foundation: Build responsive UIs for desktop applications with WPF*. Packt Publishing Ltd, 2020.

[139] Lei Zeng, Yang Xiao, Hui Chen, Bo Sun, and Wenlin Han. Computer operating system logging and security issues: a survey. *Security and Communication Networks*, 9(17):4804–4821, October 2016.

Sistemas operativos:
Una exploración y aprendizaje en Linux

En la actualidad, el empleo masivo de dispositivos electrónicos hace que los usuarios estén comprometidos a aprender cómo funcionan los sistemas operativos modernos. De hecho, reconocen que, para que funcione cualquier aplicación de software, sea web hasta un modelo de Inteligencia Artificial, debe competir por recursos computacionales limitados, tales como CPU, memoria, almacenamiento, conectividad, o seguridad entre otros. Además, sin entender cómo funciona el Sistema Operativo, un desarrollador de software no podría optimizar sus programas de computadora.

Esta obra presenta de manera didáctica los fundamentos, funciones y procesos que los sistemas operativos desempeñan. Para la aplicabilidad del conocimiento, el libro explora distribuciones de Linux y cubre actividades de aprendizaje práctico, que son desarrolladas en el laboratorio de computación o en sus máquinas virtuales.

A diferencia de los textos tradicionales, esta obra incorpora la base teórica, actividades de aprendizaje, ejercicios propuestos y resueltos, lecciones aprendidas, ejercicios de autoevaluación, entre otros que facilitan y fortalecen el aprendizaje.

Esta obra está dirigida a estudiantes universitarios, politécnicos, profesionales y público en general interesado en aprender y fortalecer su aprendizaje. Además, este libro está cimentado en la experiencia dicente y docente, profesional e investigativa de sus autores.

Finalmente, agradecemos profundamente a quienes apoyaron para la concreción de esta obra, a los directivos y autoridades de la universidad que hacen posible esta publicación, a los estudiantes de grado y posgrado que inspiraron la escritura de este libro.

Los autores

Otras obras publicadas por el autor:

Walter Fuertes & Mayra Macas: *Ciberseguridad: Del ciber-crimen a los ataques ciber-físicos* (2023) ISBN: 978-9942-765-88-8. URL: https://repositorio.espe.edu.ec/handle/21000/36481.

Walter Fuertes: *Redes de Computadoras, un Enfoque Práctico* (2022) ISBN: 978-9942-765-72-7. URL: http://repositorio.espe.edu.ec/handle/21000/30282

Fausto Meneses Becerra, Walter Fuertes: *Métodos Numéricos para profesionales en MatLab, Especialmente para Ingeniería, Economía e Informática* (2019). ISBN-13: 978-620-0-03941-5, ISBN-10: 6200039410 URL: https://www.amazon.com/M%C3%A9todos-Num%C3%A9ricos-para-Profesionales-Matlab/dp/6200039410

ISBN: 978-9942-652-56-0

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA