Metathesis:
A LaTeX template to Typeset Your Thesis for Submission to the School of Graduate Studies

(Changed the title by modifying the file `thesis.tex`*)*

Por

my-name (change this in `thesis.tex`)

Trabajo académico presentado a la

Facultad de Ingeniería

proyecto final de unidad

Master of *faculty* **or** Doctor of Philosophy (change this in `thesis.tex`)

Department of *dept-name* (change this in `thesis.tex`)

Memorial Universidad Nacional San Antonio Abad del Cusco

Month Year (change this in `thesis.tex`, too)

Cusco Perú

Proyecto de Curso:
Análisis, Diseño y Construcción de un Sistema Operativo desde Cero

Por

Castro Pari, Rayneld Fidel

Mamani Flores, Natan

Mendoza Quispe, Jose Daniel

Polo Chura, Marco Rosauro

Trabajo académico presentado a la

Facultad de Ingeniería

proyecto final de unidad

Ingeniería de Informatica y Sistemas

Departamento Académico de Ingeniería de Informatica y de Sistemas

Asesor: Ugarte Rojas, Héctor Eduardo

Memorial Universidad Nacional San Antonio Abad del Cusco

Semestre 2025-II

Cusco Perú

Abstract

This document provides information on how to write your thesis using the LaTeX document preparation system. You can use these files as a template for your own thesis, just replace the content, as necessary. You should put your real abstract here, of course.

"The purpose of the abstract, which should not exceed 150 words for a Masters' thesis or 350 words for a Doctoral thesis, is to provide sufficient information to allow potential readers to decide on relevance of the thesis. Abstracts listed in Dissertation Abstracts International or Masters' Abstracts International should contain appropriate key words and phrases designed to assist electronic searches."

— MUN School of Graduate Studies

Acknowledgements

Put your acknowledgements here...

"Intellectual and practical assistance, advice, encouragement and sources of monetary support should be acknowledged. It is appropriate to acknowledge the prior publication of any material included in the thesis either in this section or in the introductory chapter of the thesis."

— MUN School of Graduate Studies

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Marco terorico

1.1. Arquitecturas de Sistemas Operativos

1.1.1. Arquitectura Monolítica

El sistema operativo monolítico es un sistema operativo muy simple donde el núcleo controla directamente la gestión de dispositivos, memoria, archivos y procesos. Todos los recursos del sistema son accesibles al núcleo. En los sistemas monolíticos, cada componente del sistema operativo está contenido dentro del núcleo.

En una arquitectura monolítica, el núcleo del sistema operativo está diseñado para proporcionar todos los servicios del sistema operativo, incluyendo la gestión de memoria, la programación de procesos, los controladores de dispositivos y los sistemas de archivos, en un único y gran binario. Esto significa que todo el código se ejecuta en el espacio del núcleo, sin separación entre los procesos del núcleo y los del usuario (**?**).

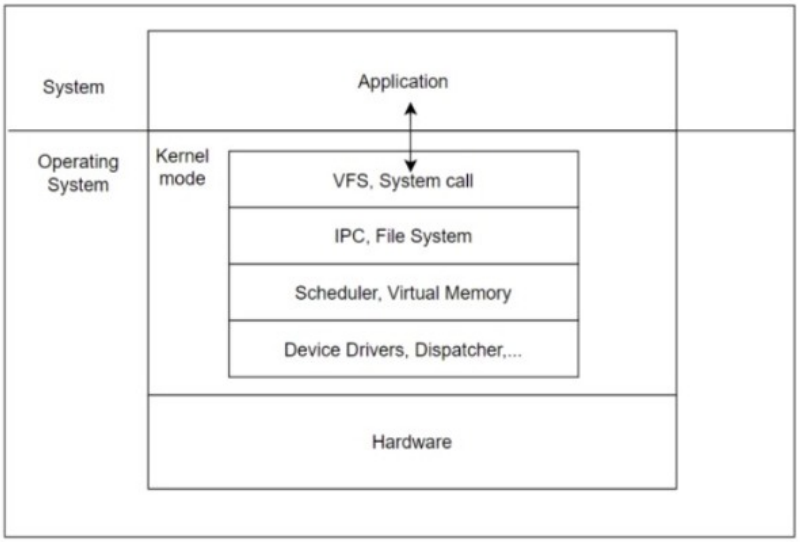

Figura 1.1: Sistema operativo basado en kernel monolítico (**?**)

1.1.2. Arquitectura en Microkernel

Un microkernel es un enfoque para diseñar un sistema operativo (SO). El microkernel proporciona servicios fundamentales para su funcionamiento, como la gestión básica de memoria, la programación de tareas.El microkernel es un tipo de sistema operativo que proporciona servicios básicos.

como los controladores de dispositivos y los sistemas de archivos, son gestionados por procesos a nivel de usuario. El proceso a nivel de usuario se comunica con el microkernel mediante el paso de mensajes. Esta forma de gestionar el proceso hace que los microkernels sean más modulares y flexibles que los kernels monolíticos tradicionales (**?**).

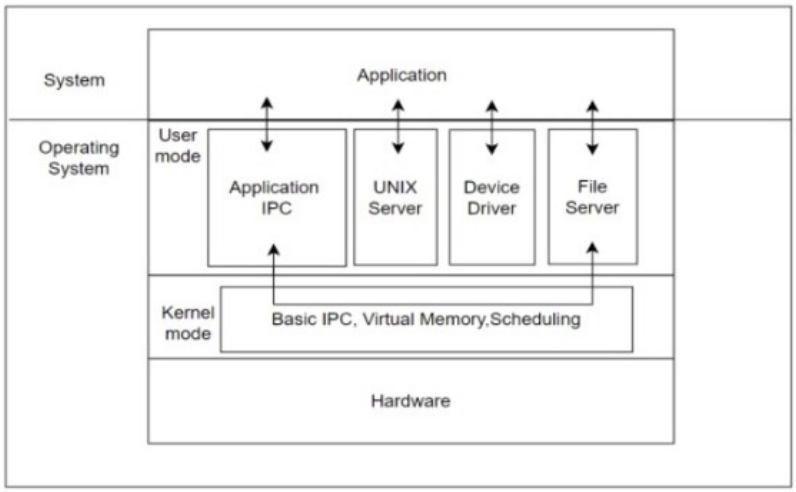

Figura 1.2: Sistema operativo basado en microkernel (**?**)

1.1.3. Arquitectura Hibrida

Un núcleo híbrido es una arquitectura de núcleo basada en la combinación de aspectos de las arquitecturas de micronúcleo y núcleo monolítico utilizadas en sistemas operativos .

La idea de esta categoría es tener una estructura de kernel similar a la de un microkernel, pero implementada en términos de un kernel monolítico. A diferencia de un microkernel, todos (o casi todos) los servicios del sistema operativo se encuentran en el espacio del kernel . Si bien no hay sobrecarga de rendimiento para el paso de mensajes ni el cambio de contexto entre el kernel y el modo de usuario, como en los kernels monolíticos , no hay ventajas de rendimiento al tener servicios en el espacio de usuario , como en los microkernels (**?**).

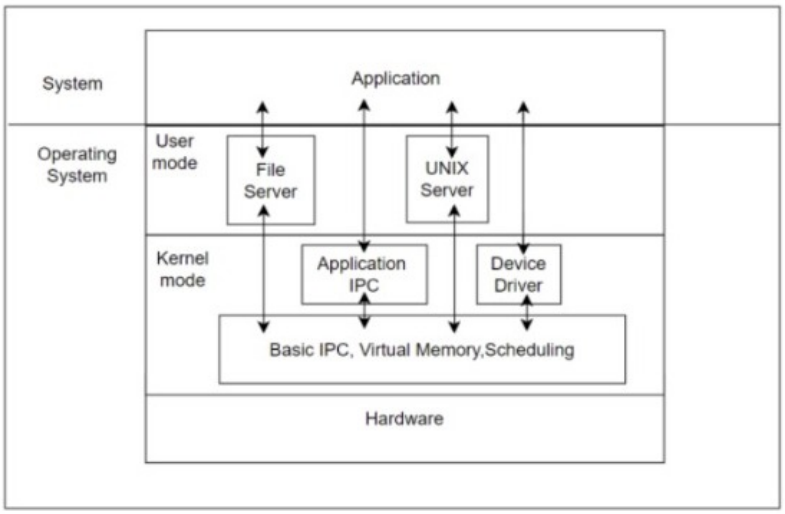

Figura 1.3: Sistema operativo basado en kernel híbrido (**?**)

1.1.4. Arquitectura en Exokernel

Exokernel es un sistema operativo desarrollado por el Instituto Tecnológico de Massachusetts (MIT) con el concepto de poner la aplicación bajo control. Los sistemas operativos Exokernel buscan proporcionar gestión de recursos de hardware a nivel de aplicación. La arquitectura de este sistema operativo está diseñada para separar la protección de recursos de la gestión, facilitando así la personalización específica de cada aplicación(**?**).

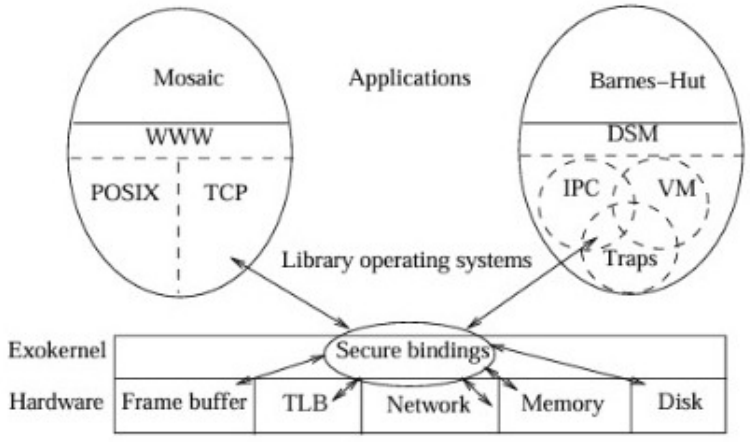

Figura 1.4: Sistema operativo basado en exokernel (**?**)

1.1.5. Comparacion entre arquitecturas de Sistemas Operativos

Cuadro 1.1: Comparación de arquitecturas de SO

Arquitectura	**Ventajas**	**Desventajas**
Monolítica	Alto rendimiento, simple en diseño inicial	Difícil de mantener, poco modular
Microkernel	Modularidad, mayor seguridad	Mayor sobrecarga de comunicación
Hibrida	Combina rendimiento y modularidad	Complejidad de implementacion
Exokernel	Máxima flexibilidad y control	Muy complejo, poco usado en producción

Capítulo 2

Componentes principales de un sistema operativo

Un sistema operativo integra varios componentes básicos que gestionan los recursos de hardware y proveen servicios a los procesos.

2.1. Gestión de procesos

Este diagrama ilustra los estados básicos por los que pasa un proceso durante su ciclo de vida. Por ejemplo, al crearse un proceso entra en el estado Nuevo; luego pasa a Preparado esperando CPU, y cuando la CPU lo atiende entra en Ejecutando. Durante la ejecución, puede quedar Esperando si necesita realizar E/S; finalmente alcanza el estado Terminado al completar su tarea, Como se muestra en la Figura

La gestión de procesos abarca funciones clave en el sistema operativo, por ejemplo la creación/destrucción de procesos, su planificación y los mecanismos de sincronización/comunicación entre ellos **?**. Entre las funciones principales se incluyen:

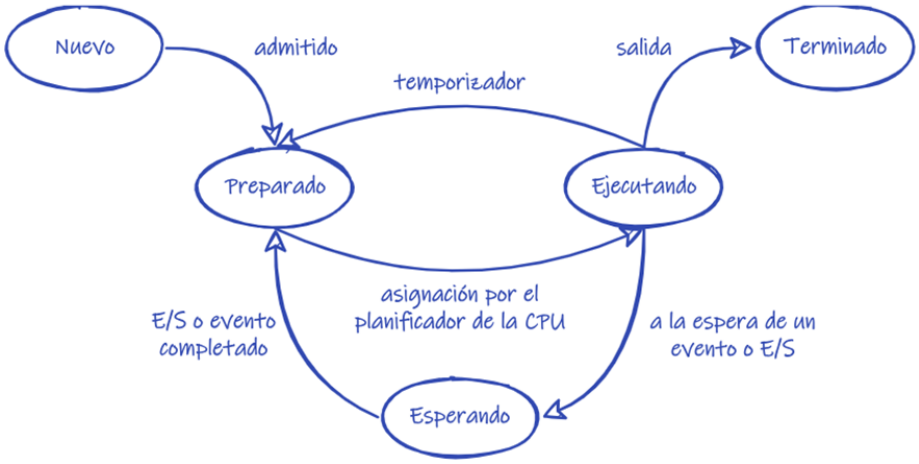

Figura 2.1: diagrama de estados de un proceso. Fuente: **?**

- **Creación y finalización**: El SO inicia nuevos procesos (por ejemplo con llamadas como `fork` en Unix) y elimina procesos completados **?**.
- **Planificación (scheduling)**: El sistema operativo decide el orden en que los procesos usan la CPU (algoritmos FIFO, round robin, prioridades, etc.) **?**.
- **Sincronización y comunicación**: Se emplean mecanismos de comunicación entre procesos (IPC) para que los procesos se comuniquen y sincronicen entre sí **?**, evitando conflictos al acceder a recursos compartidos.

Razones para la creación de un proceso

Tipo de proceso	**Descripción**
Proceso por lotes	El SO procesa trabajos en lote desde cinta/disco. Lee secuencias de control de trabajos automáticamente.
Sesión interactiva	Usuario inicia sesión desde un terminal.
Servicio del sistema	El SO crea procesos para servicios en segundo plano (ej: impresión) sin espera del usuario.
Proceso hijo	Proceso existente crea nuevos procesos para modularidad o paralelismo.

Tabla 2.1: Razones comunes para la creación de procesos en un sistema operativo. Fuente: **?**

Razones para la terminación de un proceso.

Finalización normal	Completación voluntaria mediante llamada al sistema.
Límite de tiempo	Exceso de tiempo de ejecución o espera.
Memoria no disponible	Memoria insuficiente o acceso no permitido.
Error de protección	Uso no autorizado de recursos.
Error aritmético	Operación inválida (ej: división por cero).
Fallo de E/S	Error en operación de entrada/salida.
Instrucción inválida	Ejecución de instrucción inexistente.
Instrucción privilegiada	Uso de instrucción reservada al SO.
Datos inapropiados	Datos erróneos o no inicializados.
Intervención externa	Terminación por operador o SO.
Terminación por padre	Finalización por proceso padre (automática o solicitada).

Tabla 2.2: La tabla refleja cómo los sistemas operativos gestionan y controlan la ejecución de procesos, garantizando estabilidad, seguridad y uso eficiente de los recursos. fuente: **?**

2.2. Gestión de memoria

La gestión de memoria es fundamental para que el sistema operativo asigne y libere espacio de la memoria principal (RAM) a los procesos, aislando cada espacio de direcciones y permitiendo la ejecución concurrente de múltiples programas. En esencia, el administrador de memoria asegura que la CPU pueda cargar en RAM las instrucciones y datos necesarios de cada proceso. Así, la administración de memoria organiza los procesos de modo que se obtenga la máxima utilidad del espacio disponible, trasladando la información que debe ejecutarse a memoria principal **?**,

Actualmente esta gestión se basa en técnicas de memoria virtual, que permiten al sistema disponer de una memoria lógica más amplia que la física, usando esquemas como la paginación (cada proceso ve un espacio de direcciones independiente). En la figura siguiente se ilustra conceptualmente cómo cada proceso tiene su propia tabla de páginas que mapea sus páginas virtuales (columnas) en marcos de la memoria física (rectángulos coloreados):

Entre las tareas clave del gestor de memoria se incluyen las siguientes responsabilidades:

- **Control de uso de memoria**
- **Asignación y liberación de memoria**
- **Reubicación (relocation)**
- **Fragmentación y compactación**
- **Memoria virtual**

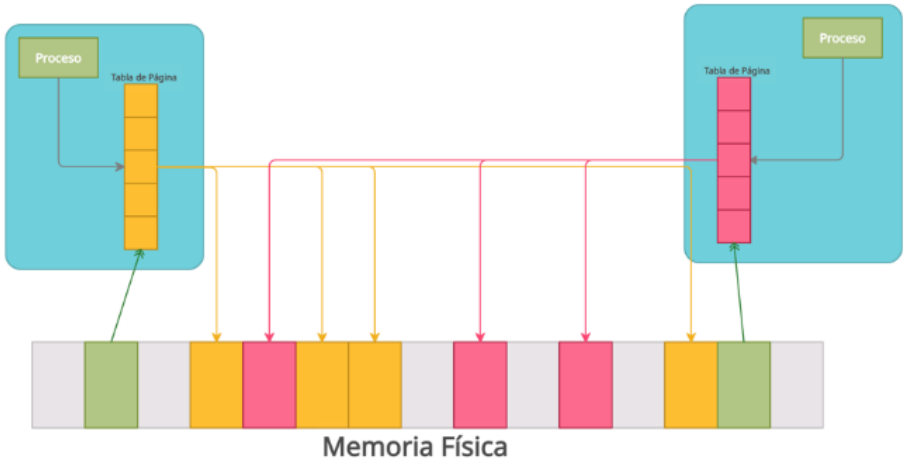

Figura 2.2: Traducción por paginación. Cada proceso (cajas turquesa) posee su propia tabla de páginas (columnas amarilla/rosada) que asigna páginas lógicas a marcos de la memoria física (fila de bloques coloreados). Fuente: **?**

- **Protección de memoria**

En conjunto, estas funciones garantizan un uso eficiente y seguro del espacio de memoria A modo de ejemplo, la figura anterior muestra cómo la traducción por paginación permite mapear páginas virtuales en marcos físicos, de modo que cada proceso "cree" tener su propio espacio contiguo, mientras que en realidad comparten la RAM de forma ordenada. El gestor de memoria actualiza las tablas de páginas y puede intercambiar páginas con el disco según sea necesario para mantener esta abstracción de memoria virtual.**?**

2.3. Sistema de archivos

2.3.1. Estructura jerárquica

El sistema de archivos organiza los archivos y directorios en forma de árbol invertido. A nivel lógico existe un único directorio raíz (/), nodo principal que contiene todos los demás. Cada nodo del árbol corresponde a un directorio que puede contener subdirectorios, archivos normales o especiales Este diseño jerárquico con nombres únicos por directorio facilita la navegación y la gestión (por ejemplo, evitando colisiones de nombres).

2.3.2. Gestión de datos persistentes

El sistema de archivos provee los medios para almacenar y recuperar datos de forma ordenada en la memoria secundaria (discos, SSD, etc.). La memoria se divide en bloques de tamaño fijo, y el sistema de archivos asigna a cada archivo los bloques necesarios.Además, controla la consistencia de los datos (por ejemplo, mediante journaling u otros esquemas) de modo que la información persista aunque finalicen los procesos o falle el sistema. También gestiona la integridad ante fallos, permitiendo recuperar estructuras de disco sin pérdida de datos. **?**(pag 1)

2.3.3. Operaciones de archivo

El sistema de archivos ofrece llamadas básicas para crear, abrir, leer, escribir, renombrar y eliminar archivos y directorios. Por ejemplo, al invocar open el kernel devuelve un descriptor de archivo, que apunta a una entrada en la tabla de archivos

abiertos y al inodo correspondiente. Según Bach (1986) esta tabla relaciona descriptores, entradas de acceso y estructuras de inodos internos. En otras palabras, cada proceso ve un descriptor (un entero), mientras que internamente el sistema mantiene una tabla de archivos que referencia los inodos de los archivos abiertos.**?**(pag. 1)

2.3.4. Asignación de espacio

Los datos de cada archivo se almacenan en bloques de disco. El sistema de archivos controla cuáles bloques están libres y asigna los necesarios a cada archivo. Por ejemplo, en el esquema FAT (File Allocation Table) cada disco tiene una tabla con una entrada por bloque; dicha entrada indica el siguiente bloque del archivo o un marcador especial (libre, defectuoso o fin de archivo)

2.3.5. Seguridad y atributos

El sistema de archivos gestiona los atributos (metadatos) de cada archivo y aplicación de permisos. Por ejemplo, en sistemas Unix cada archivo se describe con un inodo que almacena los permisos de acceso, propietario (UID/GID), fechas de modificación y tipo de archivo. **?**(pag.82)

2.3.6. Journaling

El journaling es un sistema por el cual se pueden implementar transacciones en los sistemas informáticos. También se le conoce como «registro por diario». Se basa en llevar un journal o registro de diario en el que se almacena la información necesaria para restablecer los datos afectados por la transacción en caso de que esta falle; (por

ejemplo, corte de energía) al reiniciar simplemente se «reproduce» el journal para completar o deshacer operaciones y restablecer la consistencia. de esta manera, estos sistemas de archivos se pueden restaurar a producción de forma veloz y con menos probabilidad de corrupción. **?**

2.4. Gestión de dispositivos (E/S)

El sistema de E/S del SO actúa como interfaz entre el hardware de los dispositivos (discos, teclados, impresoras, etc.) y el software, ocultando las peculiaridades de cada dispositivo al resto del sistema. **?**(seccion 4.3) Sus componentes principales son:

- **Controladores de dispositivo (drivers)**: Software específico, generalmente provisto por el fabricante, que conoce los detalles del hardware y traduce peticiones genéricas del SO en operaciones concretas sobre el dispositivo. **?**(seccion 4.3)

- **Interfaz genérica de E/S**: El SO ofrece llamadas estándar (open, read, write, close, ioctl, etc.) que los programas usan para interactuar con cualquier dispositivo sin necesitar conocer sus características físicas.(Por ejemplo, los sistemas UNIX es que todos los dispositivos de E/S se representan como un archivo en el sistema de archivos. Esto se puede comprobar rápidamente visitando el directorio /dev, permitiendo operaciones de E/S uniformes) **?**(seccion 4.3)

- **Buffering y caching**: Se enfoca en la sincronización durante una transferencia de datos en curso. Su objetivo es compensar la diferencia de velocidad durante esa operación específica (Buffering).y el caching Se enfoca en la reutilización

de datos accedidos recientemente. Su objetivo es anticiparse a futuras solicitudes, almacenando copias de datos para que los accesos subsiguientes sean ultra rápidos.

- **Spooling**: En dispositivos secuenciales no compartibles (como impresoras), el SO encola (spool) los trabajos de varios procesos en almacenamiento intermedio, gestionándolos en orden sin bloquear a los procesos remitentes.**?**(seccion 4.3)

La gestión de dispositivos garantiza así que los procesos puedan leer/escribir en hardware diverso usando una interfaz única, mientras el SO optimiza el flujo de datos y protege el acceso concurrente

2.5. Interfaz de usuario

La interfaz de usuario (IU) es el medio por el cual el usuario interactúa con el sistema.

Figura 2.3: Evolución de las interfaces de usuario - de CLI (línea de comandos) a GUI (gráfica) y a NUI (natural). Fuente: **?**

- **CLI (Command Line Interface)**: El usuario escribe comandos en una consola o terminal.

- **GUI (Graphical User Interface)**: Interfaz gráfica con ventanas, iconos y menús (más intuitiva y exploratoria)
- **NUI (Natural User Interface)**: Interacción más directa e intuitiva (por ejemplo táctil, voz o gestos).

Un buen diseño de IU busca usabilidad e intuición, permitiendo al usuario dar órdenes al SO y visualizar información de estado (p.ejemplo. terminales, escritorios, menús de configuración, iconos de carpeta, etc.). El diseño de estas interfaces puede variar ampliamente entre sistemas operativos (texto puro en ciertos sistemas Unix muy básicos, o entornos gráficos complejos en sistemas modernos)

2.6. Seguridad y protección

La seguridad en un sistema operativo garantiza que sólo usuarios autorizados utilicen los recursos del sistema y de la manera correcta Para ello se implementa el principio de menor privilegio, donde cada proceso sólo recibe los permisos estrictamente necesarios. Así, la autentificación identifica al usuario (por ejemplo, pidiendo contraseña) y le asigna permisos adecuados. Por ejemplo, en sistemas como Multics un servicio de autenticación asigna cada proceso a un usuario; si este servicio falla, un proceso podría obtener permisos no autorizados al recibir un usuario equivocado. **?**(pag. 50)

2.6.1. Control de acceso

El sistema operativo impone un modelo de control de acceso que asocia permisos con cada recurso (archivos, dispositivos, regiones de memoria, etc.) Cada solicitud (llamada al sistema) de un proceso se verifica contra estos permisos, a través de un monitor de referencia que medía y autoriza cada operación Conceptualmente, esto se puede abstraer como una matriz de acceso: una tabla donde las filas son procesos (o dominios) y las columnas objetos del sistema, y cada celda indica los permisos (leer, escribir, ejecutar, etc.) que tiene ese sujeto sobre ese objeto Por ejemplo, en Unix cada archivo tiene bits de permiso y un propietario; sólo el propietario puede cambiar los bits de permiso del archivo. **?**(pag. 27)

2.6.2. Aislamiento de procesos y protección de memoria

El SO protege la memoria y el contexto de cada proceso para impedir interferencias entre ellos. Cada proceso corre en modo usuario (no privilegiado) en su propio espacio virtual de direcciones; ningún proceso puede leer o escribir directamente la memoria de otro ni del núcleo. En particular, la protección de memoria impide que un usuario acceda a los datos de otros o modifique código/datos críticos del kernel Por ejemplo, la arquitectura de privilegios (rings) restringe instrucciones sensibles a modo kernel (nivel 0); sólo desde ese nivel privilegiado se pueden ejecutar operaciones críticas sobre memoria y dispositivos. **?**(pag. 3)

2.6.3. Prevención de intrusiones

Además de la protección interna, el SO incluye mecanismos activos contra amenazas externas o internas. Un sistema de prevención de intrusiones (IPS) supervisa el tráfico y la actividad del sistema para detectar comportamientos anómalos (por firmas conocidas o por desviaciones) y los bloquea antes de que causen daño Por ejemplo, un IPS puede terminar conexiones peligrosas o eliminar contenido maligno automáticamente Estos filtros y cortafuegos integrados en el SO actúan como barrera, rechazando accesos no autorizados (por red o por intentos locales) según políticas definidas. En conjunto, el sistema previene ataques (virus, intrusiones de red, escalada de privilegios, etc.) al observar cada petición de acceso y negarla si coincide con patrones o reglas maliciosas. **?**

2.6.4. Auditoría y registro

Para detectar incidentes y comportamientos sospechosos, el SO mantiene logs de auditoría de eventos relevantes (inicio/cierre de sesión, errores de autenticación o de acceso, cambios en permisos, etc.). Cada evento de seguridad se registra con su fecha, usuario y acción intentada. De este modo, es posible revisar los registros para identificar accesos indebidos o intrusiones tras su ocurrencia. Los sistemas de auditoría alertan automáticamente al equipo de seguridad cuando se detecta una anomalía. **?**

2.6.5. Modelos de seguridad clásicos

En los SO con seguridad reforzada también se implementan modelos formales clásicos. El modelo Bell–LaPadula es un modelo de control de acceso obligatorio

(MAC) enfocado en confidencialidad: los sujetos y objetos tienen niveles (publico, secreto, top-secret), y se aplica “no leer hacia arriba, no escribir hacia abajo” (“write up, read down”) Es decir, un sujeto sólo puede leer datos de igual o menor nivel que el suyo y escribir datos de igual o mayor nivel, evitando filtración de información sensible. En contraste, el modelo Biba protege integridad: invierte las reglas (“read up, write down”) Biba asegura que un sujeto no se contamine leyendo información de baja integridad, ni corrompa objetos de alta integridad. En la práctica estos modelos (y variantes como Clark-Wilson) definen estrictamente las relaciones de permiso para garantizar confidencialidad e integridad en sistemas multigrado. **?**

Capítulo 3

Revision de S.O. existentes

Capítulo 4

revision de sistemas operativos

4.1. Linux

4.1.1. Breve historia y proposito

Linux es un sistema operativo Unix-like de código abierto iniciado en 1991 por Linus Torvalds. Está diseñado como un núcleo (kernel) monolítico, modular y multitarea. Originalmente escrito para PCs i386, ha evolucionado hasta ofrecer compatibilidad POSIX y soporte extenso de hardware. Su desarrollo se rige por el modelo de código abierto, donde "las mejoras provienen de muchos contribuyentes corporativos e individuales" y la dirección la marca la comunidad, no un único proveedor.**? ?**

4.1.2. Arquitectura

Linux utiliza una arquitectura de kernel monolítico. Sin embargo, es importante destacar que es un kernel monolítico moderno que incorpora características de los diseños modulares. El kernel completo, incluidos todos sus controladores de dispositivos

(drivers), se ejecuta en un único espacio de direcciones en modo kernel (espacio de kernel). No obstante, soporta la carga y descarga dinámica de módulos (por ejemplo, drivers de hardware) en tiempo de ejecución, lo que proporciona una flexibilidad similar a la de los micronúcleos sin sacrificar el rendimiento de las llamadas al sistema propias de un diseño monolítico. **?**(pag. 7)

4.1.3. Lenguaje de implementacion

El núcleo de Linux está principalmente escrito en C, con algunas partes críticas en lenguaje ensamblador para optimizar el rendimiento y la interacción directa con el hardware. Las aplicaciones del espacio de usuario pueden estar escritas en cualquier lenguaje (C, C++, Python, Rust, etc.).

4.1.4. Componentes principales

- **Gestión de memoria**: Asignación y protección
- **Planificación de procesos**: Decidir qué procesos usan la CPU
- **Controladores de dispositivos**: Drivers para hardware
- **Sistema de llamadas del sistema** y **seguridad**
- **Sistema de archivos virtual (VFS)**: Unifica diversos formatos, modelos de sincronización y módulos adicionales.

?

4.1.5. Comunidad y documentación

Linux tiene una de las comunidades de código abierto más grandes y activas del mundo. Está mantenido por miles de desarrolladores de empresas (Red Hat, Google, Intel, IBM, etc.) y colaboradores individuales.

- **Documentación**: La documentación es extensa. Incluye:
    - El `man` (manual pages) integrado
    - El proyecto de documentación del kernel (`https://docs.kernel.org/`)
    - Wikis (como Arch Wiki)
    - Foros (Stack Overflow, LinuxQuestions.org)
    - Libros técnicos especializados

Capítulo 5

revision de sistemas operativos

5.1. MINIX 3

5.1.1. Breve historia y proposito

MINIX fue creado en 1987 por Andrew Tanenbaum como herramienta docente para sistemas operativos. MINIX 3 (anunciado en 2005) es una reimplementación orientada a fiabilidad y modularidad. Su objetivo es ser un SO POSIX fiable y recuperable ante fallos Es un sistema Unix-like de código abierto, escrito completamente desde cero (sin código ATT) y compatible con la versión 7 de Unix y POSIX.**?** (pag.10) Tanenbaum y colaboradores diseñaron MINIX 3 para aplicaciones críticas y educativas, enfatizando la corrección por sobre el máximo rendimiento.

5.1.2. Arquitectura

MINIX 3 emplea un microkernel muy reducido que sólo maneja interrupciones, gestión básica de procesos y comunicación IPC. Casi todo lo demás corre en espacio

de usuario como servidores independientes: controladores de dispositivos, gestor de archivos, gestor de procesos, gestor de memoria, etc. En cada servidor se confina su función ("cada servidor tiene responsabilidad limitada") y se aísla para evitar que un fallo afecte al sistema completo.Este diseño multiserver (ilustrado en la figura) permite aislar fallos: por ejemplo, el servidor de reencarnación reinicia automáticamente componentes defectuosos.**?**

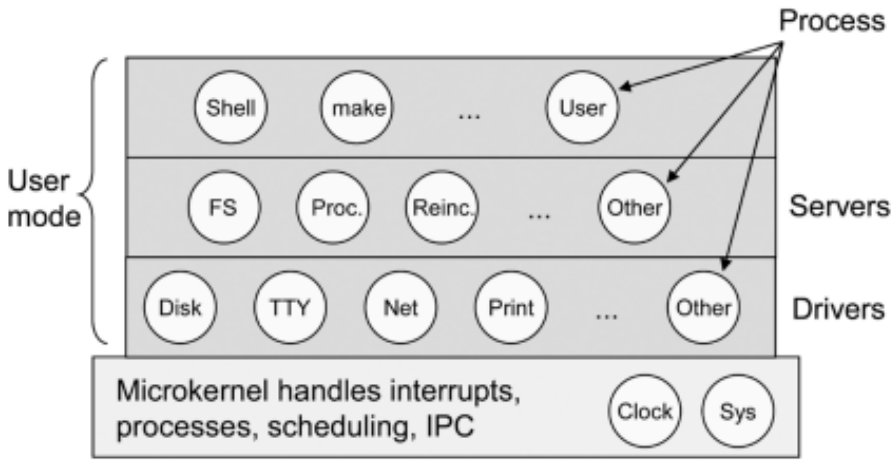

Figura 5.1: la Estructura de MINIX 3 Fuente: **?**

5.1.3. Lenguaje de implementacion

Está escrito principalmente en C, con pequeñas secciones en ensamblador para la inicialización de hardware y rutinas de bajo nivel.

5.1.4. Componentes principales

- **Gestión de procesos:** Proceso gestor supervisa creación/terminación y recolección de fallos
- **Memoria:** Microkernel provee segmentación/páginas básicas; gestor en usuario administra espacio virtual
- **Sistema de archivos:** Servidor de archivos (Mini-FS) con soporte POSIX
- **Dispositivos:** Drivers ejecutados fuera del kernel como procesos de usuario aislados
- **Interfaz de usuario:** Soporte para X11 o GUI simple (EDE) como otros Unix

?(pag. 12)

5.1.5. Comunidad y documentación

MINIX 3, aunque más pequeño, tiene presencia en la comunidad académica. Hasta 2010 contaba con ∼1.7 millones de visitas en su web oficial y 300.000 descargas de CD[1]. Ha participado en Google Summer of Code (2008–2010) y dispone de wiki, grupos de discusión y reseñas en foros. La documentación incluye el libro "Operating Systems: Design and Implementation" de Tanenbaum (3ª ed.) y artículos en congreso[2]. así como repositorios de código público (Github).

[1] `usenix.org`

[2] `https://www.inf.ufrgs.br/~johann/sisop1/minixpage.html`

revision de sistemas operativos

5.2. xv6

5.2.1. Breve historia y proposito

xv6 es un sistema operativo educativo inspirado en Unix V6, desarrollado en 2006 por el MIT para su curso de sistemas operativos. El propósito fue crear un kernel Unix simple y moderno (para x86 multiprocesador) que sustituyera a la versión 6 de Unix, de difícil manejo en la enseñanza. xv6 "sigue de cerca la estructura de Unix V6 pero está escrito en C ANSI y se ejecuta en máquinas Intel multiprocesador". **?**

5.2.2. Arquitectura

xv6 es un kernel monolítico minimalista: todo el sistema operativo principal corre con privilegios de kernel, sin separación en procesos de servidor independientes xv6 soporta multiprocesadores (usando bloqueos y la estructura de hilos del kernel) pero no provee drivers de red o gráficos avanzados. Los componentes principales incluyen: la planificación de procesos por round-robin, paginación de memoria (espacio de usuario contiguo, direcciones separadas por proceso), un sistema de archivos tipo Unix V6 sencillo, y un conjunto reducido de llamadas al sistema.**?**(pag. 25)

5.2.3. Lenguaje de implementacion

xv6 está implementado en C para la mayor parte del kernel, con una pequeña cantidad de código de inicio y manejo de interrupciones escrito en ensamblador x86

(o RISC-V en sus versiones más recientes).

5.2.4. Componentes principales

- **Gestión de procesos:** Implementa procesos con un espacio de direcciones propio. El cambio de contexto y el planificador de procesos (round-robin) son muy simples y fáciles de entender.
- **Gestión de memoria:** Utiliza una paginación simple para aislar los espacios de direcciones de los procesos. Incluye un asignador de memoria simple para el kernel.
- **Sistema de archivos:** Implementa un sistema de archivos propio e inmemorizado (inspirado en el de Unix V6) con inodos. Es un ejemplo claro de cómo se organizan los bloques, inodos y directorios.
- **Sincronización:** Introduce mecanismos de sincronización primitivos pero fundamentales como los semáforos (sleep/wakeup) para manejar la concurrencia y las condiciones de carrera.
- **Interfaces del sistema:** Expone las llamadas al sistema tradicionales de UNIX (`fork`, `exit`, `wait`, `open`, `read`, `write`, `close`, etc.), permitiendo a los estudiantes ver la implementación exacta de estas primitivas fundamentales.

?(pag. 26-30)

5.2.5. Comunidad y documentación

xv6 es ampliamente utilizado en cursos universitarios de sistemas operativos como ejemplo. Su código fuente está disponible públicamente y sus autores distribuyen un libro/comentario para enseñar sus conceptos.

Documentación: Su principal documentación es el libro 'xv6: a simple, Unix-like teaching operating system', que es un comentario línea por línea del código fuente que explica el 'qué' y el 'porqué' de cada función y estructura. El código en sí está diseñado para ser legible y está ampliamente comentado.

?

5.3. Fuchsia

Figura 5.2: Icono de S.O. de Fuchsia (**?**)

Fuchsia OS es un sistema operativo desarrollado por Google desde 2016, diseñado como una plataforma de propósito general para dispositivos embebidos, móviles, IoT y potencialmente de escritorio. A diferencia de Android, no se basa en Linux, sino en un núcleo propio denominado **Zircon** (**?**).

Su arquitectura está basada en un **microkernel**, en el que Zircon gestiona procesos, hilos, memoria y comunicación mediante objetos e IPC, mientras que servicios como sistemas de archivos y controladores se ejecutan en espacio de usuario, favoreciendo la seguridad y modularidad (**?**).

Fuchsia está escrito principalmente en **C++**, **Rust** y **Dart**. Zircon se implementa en C/C++, mientras que la capa de interfaz gráfica se desarrolla con Flutter (Dart), lo que facilita la portabilidad a múltiples plataformas (**?**).

Entre sus componentes clave se encuentran la gestión de procesos basada en objetos, memoria virtual con aislamiento estricto, soporte para múltiples sistemas de archivos (como *MemFS*, *BlobFS* y FAT) y un entorno gráfico con Flutter para experiencias multiplataforma (**?**).

El proyecto es mantenido principalmente por Google, con participación de la comunidad de código abierto. Su documentación oficial está disponible en *Fuchsia.dev*, que incluye guías técnicas, API y manuales de contribución, aunque parte del desarrollo se mantiene cerrado (**?**).

5.4. Haiku

Figura 5.3: Icono de S.O. de Haiku (**?**)

Haiku es un sistema operativo de código abierto inspirado en BeOS, iniciado en 2001 con el objetivo de proporcionar un entorno moderno, rápido y sencillo de usar, orientado principalmente a equipos de escritorio (**?**).

Su arquitectura es **híbrida**, ya que utiliza un núcleo monolítico modular, pero con elementos que recuerdan a un microkernel, lo que equilibra rendimiento y flexibilidad (**?**). El sistema está desarrollado principalmente en **C++**, con partes en C y ensamblador para las interacciones de bajo nivel (**?**).

Entre sus componentes clave se incluyen un modelo de multitarea preventiva con planificación por prioridades, gestión de memoria virtual con paginación y aislamiento de procesos, y un sistema de archivos propio llamado *OpenBFS*, optimizado para indexación rápida (**?**). Además, cuenta con una interfaz gráfica uniforme gestionada por el *app_server*.

Haiku es mantenido por la *Haiku Project Foundation* y su comunidad de desarrolladores voluntarios. El proyecto dispone de documentación oficial, foros y repositorios activos en GitHub, con lanzamientos beta periódicos que permiten probar y evaluar su madurez (**?**).

Capítulo 6

Comparación técnica

?

Bibliografía

Computer Science Department (2018). Protection and Virtual Memory. Accedido: 17 de septiembre de 2024.

Departamento de lenguajes y computacion (2005). tema 4: sistema de archivos. Accedido: 17 de septiembre de 2024.

Frans Kaashoek and Robert Morris (2016). Xv6, a simple Unix-like teaching operating system. Accedido: 16 de septiembre de 2025.

IBM (2019). ¿Qué es un sistema de prevención de intrusiones (IPS)? Accedido: 17 de septiembre de 2024.

jesustorres (2024). Sistemas Operativos. Online; accessed 20-June-2024.

jjruz (2020). tema 7: memoria virtual. Accedido: 17 de septiembre de 2024.

Jorrit N. Herder and Herbert Bos and Jakob Lichtenberg and Peter M. Chen and Andrew S. Tanenbaum (2007). The Architecture of a Reliable Operating System. Accedido: 16 de septiembre de 2024.

Juan jose costa prats (2015). gestion de procesos. Accedido: 17 de septiembre de 2024.

Love, R. (2010). *Linux Kernel Development*. Addison-Wesley Professional, third edition.

oracle latam (2024). ¿Qué es Linux? Accedido: 16 de septiembre de 2024.

rguirado (2019). tema 7: sistema de archivos. Accedido: 17 de septiembre de 2024.

Russ Cox and Frans Kaashoek and Robert Morris (2022). xv6: a simple, Unix-like teaching operating system. Accedido: 16 de septiembre de 2025.

trent jaeger (2008). operating system security. Accedido: 17 de septiembre de 2024.

usenix (2010). MINIX 3: status report and current research. Accedido: 16 de septiembre de 2024.

wikipedia (2016). Archivo:CLI-GUI-NUI, evolución de interfaces de usuario. Accedido: 17 de septiembre de 2024.

wikipedia (2022). nucleo linux. Accedido: 17 de septiembre de 2024.

Wikipedia (2023a). Gestion de memoria. Accedido: 17 de septiembre de 2024.

Wikipedia (2023b). Inter-process communication. Accedido: 17 de septiembre de 2024.

wikipedia (2023). journaling. Accedido: 17 de septiembre de 2024.

Wikipedia (2023). tabla de paginacion. Accedido: 17 de septiembre de 2024.

Apéndice A

Appendix title

This is Appendix **??**.

You can have additional appendices too (*e.g.*, `apdxb.tex`, `apdxc.tex`, *etc.*). If you don't need any appendices, delete the appendix related lines from `thesis.tex` and the file names from `Makefile`.