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

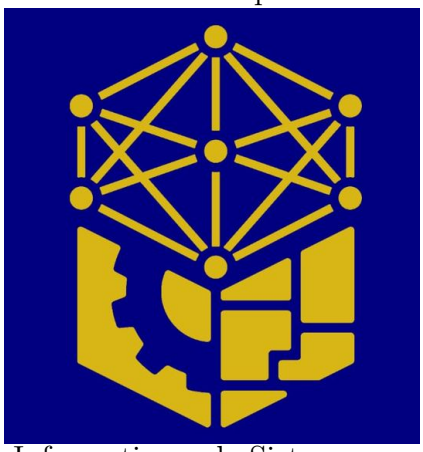
Polo Chura, Marco Rosauro

Trabajo académico presentado a la

Facultad de Ingeniería Eléctrica, Electrónica, Informática y Mecánica

como

Proyecto de Investigación de la tercera Unidad de Sistemas Operativos



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

Índice general

Índice de tablas

Índice de figuras

Introducción

El desarrollo de sistemas operativos constituye uno de los campos más desafiantes y fundamentales de la ingeniería informática. Comprender los mecanismos internos que permiten la gestión de recursos de hardware, la planificación de procesos, la administración de memoria y la comunicación entre componentes del sistema resulta esencial para cualquier profesional que aspire a dominar los fundamentos de la computación moderna. En este contexto, el presente proyecto de investigación tiene como propósito analizar, diseñar y documentar la construcción de un sistema operativo desde cero, utilizando como caso de estudio el sistema operativo HelenOS.

HelenOS es un sistema operativo de código abierto basado en una arquitectura de micronúcleo multiservidor, desarrollado originalmente por Jakub Jermář en 2001 como un proyecto educativo y de investigación (Děcký, 2015). A diferencia de los sistemas monolíticos tradicionales, HelenOS implementa un diseño donde el núcleo se mantiene mínimo y las funcionalidades del sistema se ejecutan como servidores independientes en espacio de usuario, comunicándose mediante mecanismos de paso de mensajes (IPC). Esta arquitectura modular facilita el análisis individual de cada componente y permite una comprensión más clara de las interacciones entre los diferentes subsistemas.

Motivación

La elección de HelenOS como base para este proyecto responde a múltiples factores técnicos y pedagógicos. En primer lugar, su arquitectura microkernel multiservidor representa un enfoque moderno y elegante para el diseño de sistemas operativos, permitiendo aislar fallos y mantener una separación clara entre mecanismos y políticas. En segundo lugar, el proyecto cuenta con una base de código bien estructurada, documentación accesible y una comunidad académica activa que lo respalda. Finalmente, HelenOS ofrece un equilibrio adecuado entre complejidad y accesibilidad, siendo más sofisticado que sistemas puramente didácticos como Minix o xv6, pero sin alcanzar la complejidad abrumadora de sistemas de producción como Linux.

Objetivos del proyecto

El objetivo general de este proyecto es realizar un análisis integral del sistema operativo HelenOS, comprendiendo su diseño arquitectónico, sus componentes principales y los mecanismos que lo hacen funcionar. Los objetivos específicos incluyen:

- Justificar técnica y pedagógicamente la selección de HelenOS como sistema operativo base para el estudio.
- Analizar la arquitectura microkernel multiservidor y sus ventajas frente a diseños monolíticos.
- Estudiar los componentes fundamentales del sistema: bootloader, kernel, gestión de procesos, gestión de memoria, sistema de archivos e interfaz de usuario.

- Documentar las herramientas y el entorno de desarrollo necesarios para trabajar con HelenOS.
- Elaborar un plan de implementación que permita extender o modificar componentes del sistema de manera progresiva.
- Identificar riesgos potenciales y estrategias de mitigación para el desarrollo del proyecto.

Alcance

El presente documento abarca el análisis teórico y el diseño conceptual del sistema operativo HelenOS, correspondiente a la tercera unidad del curso de Sistemas Operativos. El alcance incluye la documentación detallada de la arquitectura del sistema, sus componentes principales, las políticas de planificación y manejo de recursos, así como la planificación para una implementación incremental. No se incluye en este documento la implementación completa del código, sino el análisis y diseño que servirán como base para futuras etapas de desarrollo.

Estructura del documento

El documento está organizado en seis capítulos principales, además de esta introducción:

- **Capítulo 1: Justificación de la propuesta seleccionada.** Presenta los argumentos que fundamentan la elección de HelenOS, incluyendo sus objetivos originales, arquitectura, nivel de complejidad y recursos de soporte disponibles.

- **Capítulo 2: Argumentos técnicos.** Detalla las características técnicas que hacen de HelenOS una opción viable, como su compatibilidad con herramientas de desarrollo, modularidad, lenguaje de programación y facilidad de compilación.

- **Capítulo 3: Argumentos pedagógicos.** Expone las razones educativas para elegir HelenOS, destacando su claridad conceptual, potencial para el aprendizaje activo y relación con los contenidos del curso.

- **Capítulo 4: Diseño del sistema operativo propuesto.** Presenta el diseño detallado de HelenOS, incluyendo diagramas de arquitectura, componentes a implementar, políticas de planificación y flujo de ejecución.

- **Capítulo 5: Herramientas y entorno de desarrollo.** Describe las herramientas necesarias para el desarrollo, incluyendo compiladores, emuladores, control de versiones y editores de código.

- **Capítulo 6: Planificación de implementación.** Presenta el cronograma tentativo, la estrategia de pruebas y validación, y el análisis de riesgos del proyecto.

Metodología

Para el desarrollo de este proyecto se adoptó la metodología ágil Scrum, adaptada al contexto académico del curso. El trabajo se organizó en sprints semanales, permitiendo entregas incrementales de documentación e investigación. Esta metodología

facilitó la distribución de tareas entre los integrantes del equipo, aprovechando la naturaleza modular de HelenOS para asignar componentes específicos a cada miembro. Las revisiones periódicas permitieron mantener la calidad del trabajo y realizar ajustes según las necesidades identificadas durante el proceso de investigación (SCRUMstudy, 2022).

La investigación se basó en fuentes primarias, incluyendo la documentación oficial de HelenOS, tesis de grado y maestría relacionadas con el proyecto, artículos académicos publicados por los desarrolladores principales y el análisis directo del código fuente disponible en el repositorio oficial de GitHub (Jermář, 2025b).

Capítulo 1

Justificación de la propuesta seleccionada

1.1 Nombre del sistema operativo base

El nombre del sistema operativo es "HelenOS". El proyecto fue iniciado por Jakub Jermar en 2001, en ese momento era un código independiente que funcionaba como kernel para IA-32. (Děcký, 2015)

1.2 Objetivos del proyecto original

Según (Děcký, 2015), este sistema operativo, tiene un enfoque educativo y experimental. Además, busca servir como una plataforma para la investigación y desarrollo de sistemas operativos de propósito general, teniendo muy en cuenta la fiabilidad y practicidad. De manera más organizada se muestran los siguientes objetivos principales en forma de lista:

- Proporcionar una plataforma real para el estudio y la experimentación en el diseño de sistemas operativos modernos.
- Implementar una arquitectura microkernel funcional que priorice la seguridad, la modularidad y la fiabilidad.
- Facilitar el aprendizaje de conceptos fundamentales de sistemas operativos mediante una implementación clara, accesible y extensible.
- Servir como base para el desarrollo incremental de servicios del sistema, controladores de dispositivos y mecanismos de comunicación entre procesos.

1.3 Arquitectura y enfoque técnico

El sistema operativo HelenOS se basó en la arquitectura multiserver de microkernel, donde el kernel de encarga de tareas muy específicas y principal; y los servicios son implementados en componentes aislados e independientes, en esta caso implementados con permisos de modo privilegiado para el kernel y modo usuario los demás (Děcký, 2015). A continuación, se incluye una vista general de la arquitectura/organización del kernel de HelenOS:

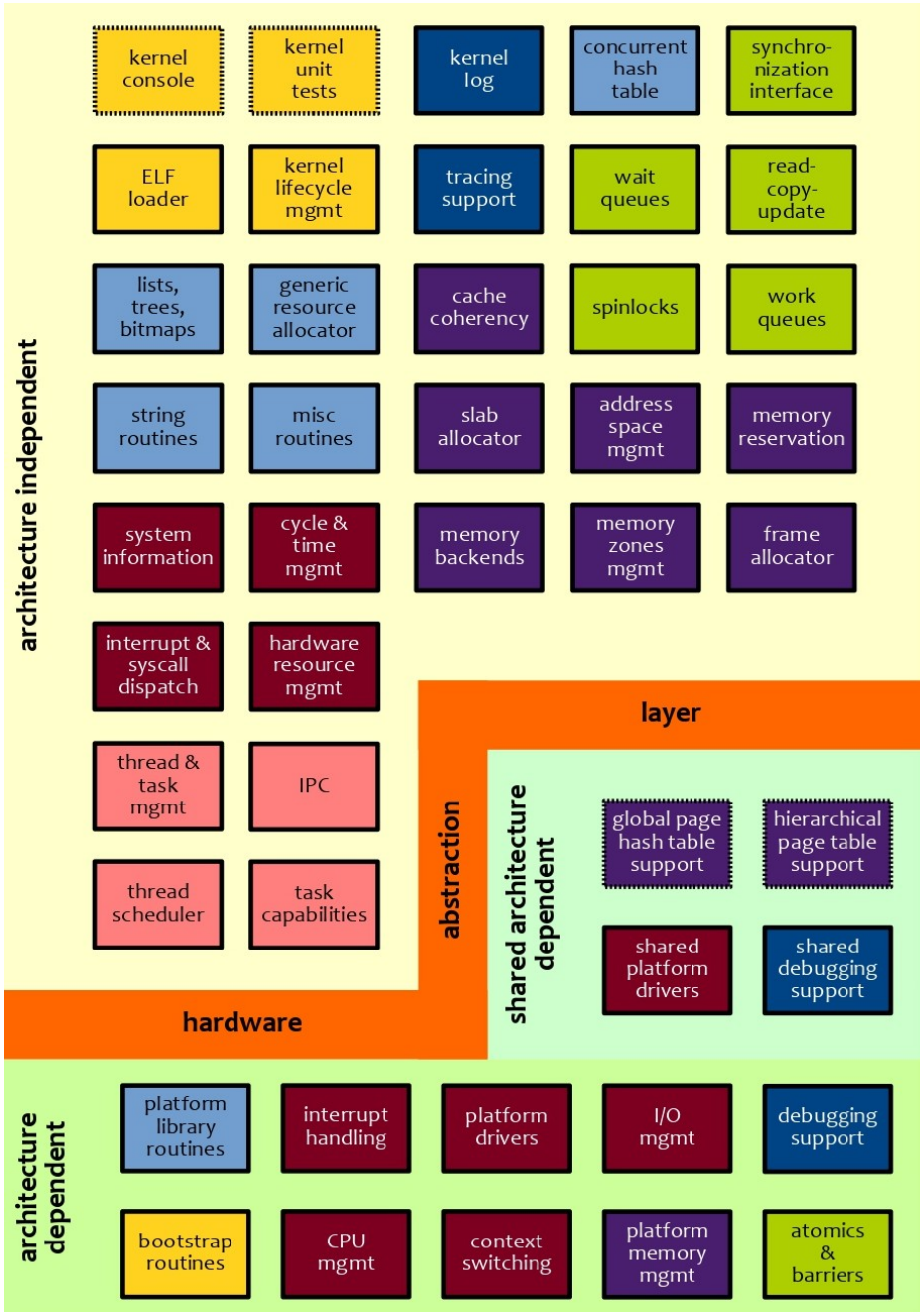


Figura 1.1: Vista general de la arquitectura/organización del kernel de HelenOS.

Fuente: Obtenido de (Děcký, 2015) *Application of Software Components in Operating System Design.*

El enfoque técnico es claro, al haber iniciado como un proyecto educativo independiente, se enfoca en la simplicidad y claridad del código, manteniendo la modularidad respectiva a su arquitectura elegida. En principio, HelenOS fue diseñado para la arquitectura IA-32, pero al día de hoy soporta múltiples arquitecturas, incluyendo x86-64, ARM y MIPS (Děcký, 2015).

1.4 Nivel de complejidad y adecuación al contexto educativo

Considerando un entorno de estudio de sistemas operativos, justamente el entorno en el que se propicia esta investigación, con conocimientos previos de C y fundamentos de sistemas operativos; es un proyecto con un nivel de complejidad media, más que nada por su tamaño (es extenso), además del uso de servicios mediante IPC y también considerar que la arquitectura multiserver, que implica microkernel, no es la más sencilla tampoco. Y eso sin contar la diversidad de arquitecturas soportadas, lo cual añade dificultad al análisis del código.

Se escogió este sistema operativo debido a que es bastante adecuado para este contexto educativo, ya que su diseño modular permite su análisis en forma progresiva, empezando por componentes básicos y terminando en los más complejos. Además, otras de las razones para decir que es adecuado son su excelente estructura del código, la separación clarísima de sus componentes y, con más peso para este proyecto, su escalabilidad ya que podemos implementar de manera gradual más componentes de manera que se integre al sistema completamente.

1.5 Comunidad, documentacion y soporte disponible

Se consideró en la categorización del nivel de complejidad que, la accesibilidad para los estudiantes es bastante buena, ya que el proyecto es open source y cuenta con una gran cantidad de estudios en torno a él. Sin embargo, una consideración a tomar en cuenta es que, aunque HelenOS dispone de documentación formal (guías, artículos, documentación generada), esta puede no detallar cada componente del sistema operativo con la exhaustividad que otros sistemas operativos grandes ofrecen (por ejemplo un libro completo o wiki ultra-detallada), aunque esta deficiencia se ve compensada por la claridad del código y la estructura modular, sobre su comunidad es importante decir que no es muy grande, sin embargo es activamente estudiado a nivel académico y cuenta con foros donde se puede solicitar ayuda, en algunas redes sociales como reddit y GitHub.

Capítulo 2

Argumentos técnicos

2.1 Compatibilidad con herramientas de desarrollo

HelenOS es compatible con herramientas de desarrollo muy populares, además de contar con un repositorio en github, por lo que para poder clonarlo se puede usar git y para posteriormente modificarlo se puede usar cualquier editor de código como Visual Studio Code. Por otro lado, para su proceso de construcción, compilación y ejecución, se usan herramientas estándar de GNU incluyendo sus compiladores GCC mediante configuraciones de compilación cruzada; también para ejecutarse y probarse se puede usar emuladores y máquinas virtuales como QEMU, VirtualBox y más (Jermář, 2025b). Esta razón de la facilidad de uso de las herramientas que se pueden usar para el proceso de desarrollo, es un gran argumento a favor para escoger este sistema operativo como caso de estudio, ya que se puede dedicar más tiempo a analizar el sistema en sí en vez de sus herramientas.

2.2 Modularidad y posibilidad de adaptación

Como se viene repitiendo a lo largo del presente documento, la modularidad es el fuerte de HelenOS, gracias a su arquitectura microkernel multiserver, es su argumento más fuerte a favor para escogerlo, una modularidad muy bien implementada, organizada y documentada. Y es justamente gracias a esta excelente implementación de modularidad que se disfrutan de todos sus beneficios, como un fácil reemplazo, agregación o adaptación de cualquier componente o servicio, sin afectar el kernel del sistema; el permiso de ver cada componente como una caja negra para facilitar su análisis exploratorio y sobre todo el beneficio principal para este proyecto, la escalabilidad. Todos esos beneficios ayudan a planificar el trabajo en equipo, al asignar diferentes módulos a cada integrante, además de un desarrollo incremental sobre el que se puede aplicar metodolgías ágiles.

2.3 Lenguaje de programación utilizado

Este sistema está implementado en el lenguaje de programación C, que como ya se mencionó se asume como conocimiento previo a este curso, por lo que también fue un argumento a favor para su elección, ya que no sería demasiada la carga para analizar el código, más que repasar conceptos del lenguaje; además de que C aunque no fuese un conocimiento previo, es un lenguaje ampliamente utilizado para sistemas operativos debido a que permite un manejo de memoria y recursos del sistema muy preciso. También se usa ensamblador pero en mucha menor medida.

2.4 Facilidad de compilación, prueba y depuración

Sobre el proceso de compilación de HelenOS, este se encuentra bastante bien documentado y se basa en makefiles para generar el sistema operativo completo mediante comandos estándar. Además, como se mencionó en anteriores secciones, para su ejecución se pueden usar emuladores o máquinas virtuales, simplificando así el proceso de prueba y ejecución. En cuanto a la depuración, HelenOS proporciona sus propios mecanismos como la salida de información por consola, soporte para símbolos de depuración y poder observar el comportamiento de los servicios en tiempo real y de manera independiente con herramientas adicionales implementadas por los mismos desarrolladores en un repositorio adicional (Jermář, 2025a).

2.5 Escalabilidad para futuras extensiones

Como se mencionó en la sección de modularidad, la escalabilidad es uno de los beneficios más claros de HelenOS, permite la incorporación de nuevas funcionalidades/componentes de manera progresiva sin afectar a la estabilidad del kernel. Sin embargo, se debe tener en cuenta que para estas incorporaciones se debe seguir aplicando la arquitectura microkernel multiserver, además de la filosofía modular y los diferentes mecanismos para mantenerla, tales como la comunicación entre processo (IPC) en caso se necesite. Esta alta capacidad de escalabilidad es un argumento muy fuerte a favor de HelenOS para ser escogido como caso de estudio para este proyecto en específico. Este hecho quedó demostrado en una gráfica hecha por los autores para medir la cantidad de líneas de código que tuvo el proyecto a lo largo del tiempo:

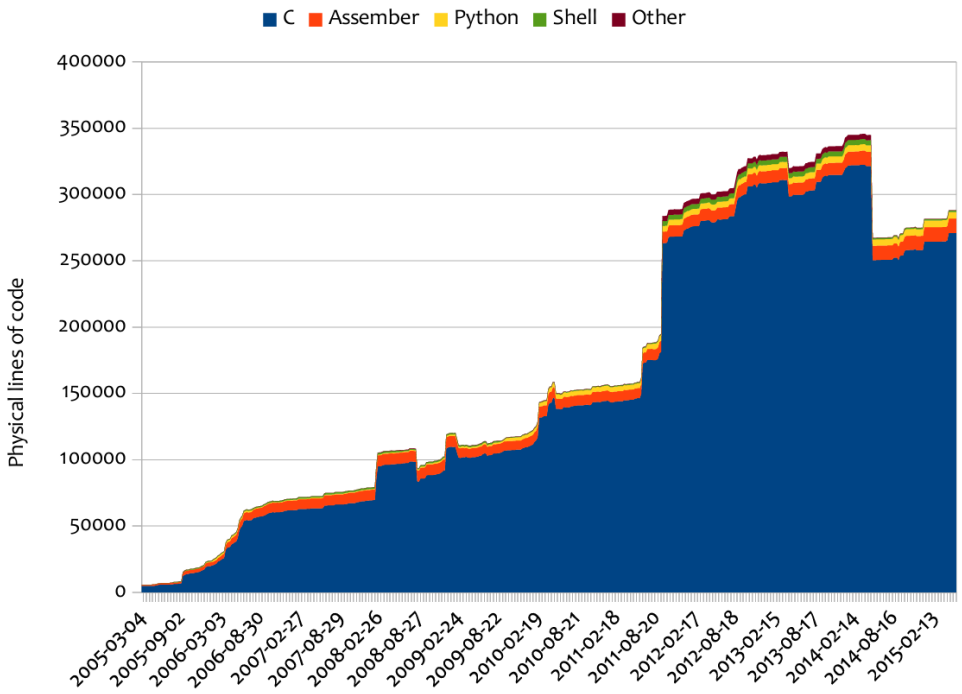


Figura 2.1: Escalabilidad en líneas de código del respositorio de HelenOS a lo largo del tiempo (fuente: (Děcký, 2015)).

Capítulo 3

Argumentos pedagógicos

3.1 Claridad conceptual y didáctica

HelenOS tiene una arquitectura bien clara y definida, con su organización en componentes y módulos favorece la compración de los conceptos fundamentales de los sistemas operativos. HelenOS muestra de manera directa mecanismos para mantener la modularidad como la comunicación entre procesos (IPC), planificación de hilos y gestión de memoria; facilitando y clarificando aún más el análisis conceptual del sistema operativo en profundidad. En conclusión es un sistema operativo muy didáctico y con mucho potencial para ser usado en la enseñanza de los cursos de sistemas operativos.

3.2 Potencial para fomentar el aprendizaje activo

El uso de este sistema operativo como base del proyecto apoya y contribuye al aprendizaje activo, ya que los estudiantes participaremos directamente en la contrucción,

modificación, agregación y prueba de un sistema operativo real. El proceso de desarrollo implica proactividad, para aprender a configurar y compilar el sistema, implica también análisis del flujo de ejecución y componentes individuales del sistema operativo. Además, al implementar nuevas funcionalidades por cuenta propia, se puede llegar a cometer errores cuya solución refuerza el conocimiento adquirido a partir de este aprendizaje experimental. En específico, este sistema operativo tiene un montón de potencial para fomentar el aprendizaje activo sobre todo si se quiere enseñar algo más moderno y aplicable actualmente, a diferencia de opciones más sencillas como Minix o XV6 al contar con arquitecturas muy sencillas o limitadas, así como también diferente a opciones más complejas que pueden llevar a una parálisis por sobrecarga de información, como Theseus o Linux. Razones por las cuales se escogió HelenOS al considerarlo un equilibro adecuado.

3.3 Relación con los contenidos del curso

El sistema operativo HelenOS está muy bien alineado a los contenidos del curso específico de la investigación, incluyendo en su código todos los temas avanzados (como planificación de procesos e hilos, gestión de memoria, comunicación entre procesos (IPC), sistemas de archivos y la gestión de E/S) varios de ellos con incluso más profundidad de la que nos permitió el avance del curso; así como también conceptos relativos a sistemas operativos modernos, tales como la gestión de drivers como un árbol y soporte para múltiples tipos de buses. Se asegura un correspondencia casi perfecta entre la teoría impartida en el curso y su implementación práctica en HelenOS, permitiendo así que los estudiantes relacionemos los conceptos abstractos con

soluciones concretas.

3.4 Posibilidad de trabajo colaborativo y evaluación progresiva

En HelenOS y gracias a su estructura modular, es fácilmente aplicable un trabajo en equipo y una planificación en varias etapas. En cada etapa, cada componente independiente puede ser asignado a uno de los integrantes del equipo; queremos resaltar que identificamos que el proyecto era altamente compatible para aplicar la metodología ágil Scrum, aplicando sus principios, asegurando así un trabajo en equipo efectivo, una evaluación de calidad constante y progresiva, y en general un buen desarrollo del proyecto; se aplicó de forma que las etapas identificadas en el cronograma fueron tratadas como sprints, y se aseguró calidad al entregarse incrementos de documentación e implementación al final de cada sprint (SCRUMstudy, 2022). En esencia, HelenOS posee una altísima posibilidad de trabajo en equipo y evaluaión progresiva debido en gran medida a su arquitectura microkernel multiserver y su excelente organización modular.

Capítulo 4

Diseño del sistema operativo propuesto

Este capítulo presenta el diseño del sistema operativo HelenOS, Diseñado a partir de una arquitectura de micronúcleo y entornos operativos multiservidor (Děcký, 2015, 2010; Jermář, 2025b). Se detallan el diseño estructural y los módulos esenciales para el desarrollo, las políticas de planificación y manejo de recursos y el flujo de ejecución esperado desde el arranque hasta la interacción del usuario. Se implementa un núcleo básico apoyado por múltiples servidores que operan en el área de usuario como redes, archivos y controladores, comunicándose mediante paso de mensajes asíncrono (Děcký, 2015). Ademas se inspirado en Unix/POSIX, evita interfaces heredadas cuando existen alternativas modernas, como por ejemplo, prescinde de los sockets de POSIX y expone una API orientada a flujos TCP (Korop, 2025; HelenOS project, 2026). La ruta de la comunicación de red atraviesa por procesos (NIC $\rightarrow$ Ethernet $\rightarrow$ IP $\rightarrow$ TCP) antes de llegar a la aplicación, reforzando el aislamiento y la modularidad (Ko-

rop, 2025). Finalmente, los ejecutables siguen el formato ELF con soporte de enlace dinámico, PIE y TLS, preparados por un servidor de carga antes del inicio de cada tarea (Volf, Matěj, 2025).

4.1 Diagrama de arquitectura general

El sistema operativo HelenOS está diseñado como un microkernel relativamente pequeño, asistido por un conjunto de controladores de espacio de usuario y tareas de servidor. HelenOS no es muy radical en cuanto a qué subsistemas deben o no implementarse en el kernel, en algunos casos, existen tanto controladores de kernel como de espacio de usuario. La razón para crear el sistema como un microkernel es prosaica. Si bien inicialmente es más difícil obtener el mismo nivel de funcionalidad de un microkernel que en el caso de un kernel monolítico simple, un microkernel es mucho más fácil de mantener una vez que sus componentes se han puesto en funcionamiento. Por lo tanto, el kernel de HelenOS, así como sus bibliotecas esenciales de espacio de usuario, solo pueden ser mantenidas por unos pocos desarrolladores que las comprendan completamente. Además, un sistema operativo basado en microkernel se completa antes que los kernels monolíticos, ya que el sistema puede utilizarse incluso sin algunos subsistemas tradicionales (por ejemplo, dispositivos de bloque, sistemas de archivos y redes). Segun (HelenOS Project, 2006).

Pordemos observar el diagrama de arquitectura general en la Figura 4.1.

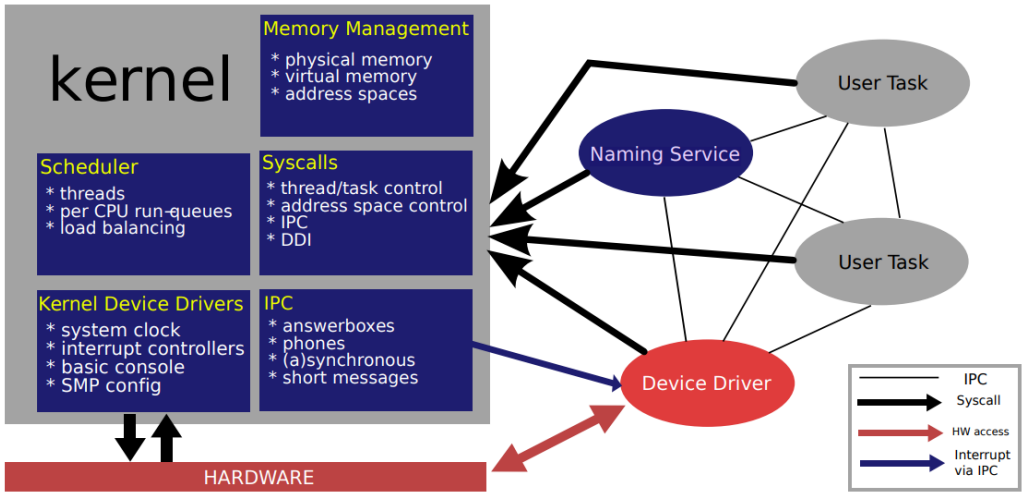


Figura 4.1: Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2006)).

Además de compararla con otras:

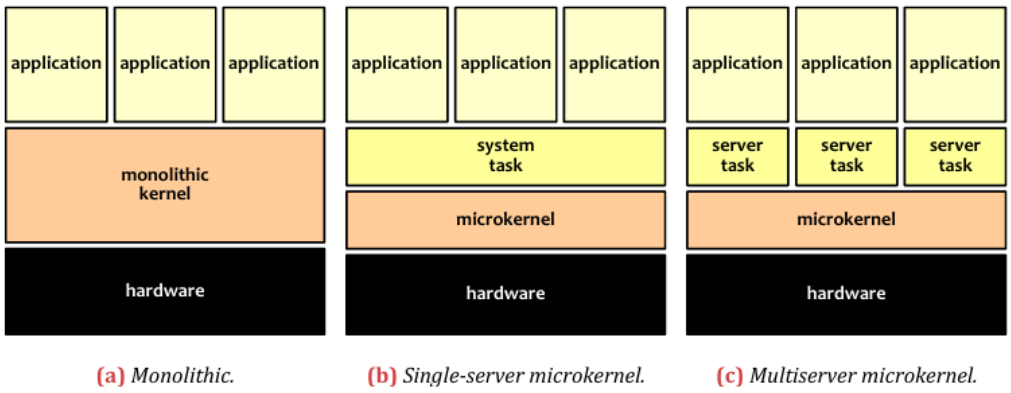


Figura 4.2: Comparación con arquitecturas clásicas (fuente: (Děcký, 2015)).

4.2 Componentes a implementar

4.2.1 Bootloader

En la arquitectura x86-64, HelenOS emplea el gestor de arranque GRUB (Grub boot loader). Segun (HelenOS Project, 2026) para iniciar el sistema. Este se encarga de cargar el kernel junto con un conjunto inicial de tareas de espacio de usuario necesarias para completar el proceso de arranque. Asimismo, GRUB carga un disco RAM que contiene el sistema de archivos raíz. Durante las fases iniciales del arranque, el sistema muestra mensajes de registro generados tanto por el kernel como por las tareas de espacio de usuario a medida que se inicializan. Una vez completado este proceso, el compositor de la interfaz gráfica toma control de la pantalla, momento en el cual el sistema ya cuenta con más de 35 tareas de espacio de usuario en ejecución, responsables de proporcionar la funcionalidad básica del sistema. Según la documentación de diseño y del propio proyectos (HelenOS Project, 2006; Jermář, Jakub and Děcký, Martin and Cejpek, Josef and Mejdrech, Lukáš and others, 2006).

4.2.2 Kernel básico

Segun (Děcký, 2010)El planificador de HelenOS mantiene varias colas de ejecución asociadas a cada procesador. Los hilos preparados para ejecutarse se insertan en estas colas de acuerdo con su nivel de prioridad y el procesador actual, desde donde son seleccionados para su ejecución. Para garantizar un reparto equilibrado de la carga, existen hilos del núcleo con funciones especiales encargados de migrar hilos entre procesadores cuando es necesario. La planificación se basa en una política de

round-robin aplicada sobre múltiples colas de prioridad.

Aunque el diseño del micronúcleo prioriza la simplicidad conceptual, HelenOS hace uso de mecanismos modernos y eficientes, incluyendo árboles AVL y B+, tablas hash, un asignador de memoria SLAB, con el objetivo de mejorar el rendimiento y la escalabilidad del sistema.

El microkernel de HelenOS está organizado en subsistemas especializados que gestionan los recursos esenciales del sistema. La Tabla 4.1 presenta los tres pilares fundamentales: programación, gestión de memoria y comunicación entre procesos, detallando los mecanismos clave que implementa cada uno (HelenOS Project, 2006; Děcký, 2010).

Tabla 4.1: Subsistemas fundamentales del kernel de HelenOS

Subsistema del Kernel	**Entidades / Mecanismos**	**Función Principal**
Programación (Scheduling)	Hilos del kernel, hilos de usuario, pseudohilos, tareas, colas de ejecución multinivel, planificador round-robin con prioridades, balanceo de carga entre CPUs	Gestionar la ejecución concurrente de múltiples tareas, asignar tiempo de CPU según prioridad, mantener el equilibrio de carga entre procesadores y soportar sistemas multiprocesador (SMP)

Tabla 4.1 – continuación de la página anterior

Subsistema del Kernel	**Entidades / Mecanismos**	**Función Principal**
Gestión de Memoria	Asignador de bloques, asignador de tramas físicas (buddy system), traducción virtual-física mediante tablas de páginas, espacios de direcciones aislados, áreas de memoria compartidas, coherencia TLB	Asignar memoria dinámica al kernel y a las tareas de usuario, traducir direcciones virtuales a físicas, aislar espacios de memoria entre procesos, gestionar memoria compartida y garantizar coherencia en sistemas multiprocesador
Comunicación entre Procesos (IPC)	Mensajes cortos síncronos/asíncronos, *teléfonos* (canales de comunicación), llamadas asíncronas con callbacks, cuadros de respuesta, áreas de memoria compartida para datos grandes	Permitir la comunicación eficiente y segura entre tareas del sistema en un entorno microkernel multiservidor, facilitando el paso de mensajes y compartición de datos entre procesos aislados

Esta arquitectura modular permite que el microkernel se mantenga compacto y simple, delegando funcionalidades complejas a servidores en espacio de usuario que se comunican mediante el sistema IPC del kernel.

4.2.3 Gestión de procesos

En HelenOS, el hilo es la unidad básica de ejecución del kernel y se agrupa en tareas según su funcionalidad. En el espacio de usuario se emplean fibrillas, hilos cooperativos construidos sobre una API del núcleo y utilizados por el marco asíncrono. Debido a su arquitectura de micronúcleo, la comunicación entre procesos (IPC) es fundamental. Las tareas intercambian información mediante mensajes breves o mediante el compartimiento de memoria para datos de mayor tamaño. El modelo de IPC permite múltiples conexiones simultáneas entre tareas. (HelenOS Project, 2006; Jindrák, Jaroslav, 2022).

Aspectos esenciales del manejo de hilos y tareas

Hilos y tareas. Una tarea representa un espacio de direcciones y recursos, mientras que el hilo es la unidad básica de ejecución. Cada tarea se inicia con un hilo principal creado por el cargador y registrado en el planificador del sistema.

Planificación y ejecución. Los hilos pasan por los estados *listo*, *ejecutando* y *bloqueado*. El planificador selecciona el hilo a ejecutar según prioridades y políticas de reparto justo, utilizando preempción basada en temporizadores.

Conmutación de contexto. La conmutación de contexto ocurre ante interrupciones, llamadas al sistema o bloqueos. El kernel guarda el estado del hilo actual y restaura el del siguiente, permitiendo la ejecución concurrente eficiente.

Comunicación entre procesos (IPC). El modelo cliente–servidor permite la comunicación mediante mensajes y, cuando es necesario, memoria compartida. El cliente envía una solicitud, el servidor la procesa y responde, desbloqueando al cliente.

Creación y finalización. El cargador valida y carga el ejecutable, crea las áreas de memoria y el hilo inicial. Al finalizar un hilo, se liberan sus recursos; si la tarea queda sin hilos activos, el sistema libera completamente la tarea.

4.2.4 Gestión de memoria

HelenOS garantiza la coherencia entre la TLB y las tablas de páginas mediante un mecanismo de invalidación coordinada en sistemas multiprocesador. La gestión de memoria del sistema cubre la asignación para el kernel, la traducción de direcciones virtuales y la administración de espacios de direcciones. Cada espacio de direcciones está compuesto por áreas disjuntas respaldadas por memoria anónima, imágenes ejecutables o memoria física. El sistema permite compartir áreas entre tareas, pero no soporta intercambio de páginas con almacenamiento secundario. (HelenOS Project, 2006)

Tabla 4.2: Aspectos esenciales de la gestión de memoria en HelenOS

Componente	**Descripción técnica**	**Objetivo / Beneficio**
Zonas y marcos físicos	La memoria física se organiza en zonas contiguas que contienen marcos (*frames*) administrados mediante un *buddy system*. Cada zona mantiene contadores de marcos libres y ocupados.	Permitir asignación eficiente de bloques contiguos y escalabilidad del sistema de memoria física.

Tabla 4.2 – continuación

Componente	**Descripción técnica**	**Objetivo / Beneficio**
Asignador de marcos	El *frame allocator* gestiona solicitudes de memoria en potencias de dos dentro de una zona, utilizando el *buddy allocator*.	Asignación rápida y consistente de memoria física alineada.
Slab allocator	Asignador de objetos pequeños y frecuentes basado en *slabs* y *magazines* por CPU, evitando bloqueos globales.	Reducir fragmentación interna y mejorar concurrencia y rendimiento del kernel.
Memoria virtual	HelenOS implementa paginación mediante una interfaz genérica que soporta tablas jerárquicas o tabla hash global según la arquitectura.	Abstraer diferencias de hardware y unificar la gestión de memoria virtual.
Espacios de direcciones	Cada tarea posee un espacio de direcciones con áreas disjuntas organizadas en un B+ tree; el kernel se mapea fuera de estas áreas.	Aislamiento entre procesos y gestión independiente de regiones de memoria.
Áreas y permisos	Las áreas definen regiones homogéneas con permisos de lectura, escritura y ejecución, evitando combinaciones inseguras como WRITE+EXEC.	Control fino de protección y seguridad de memoria.

Tabla 4.2 – continuación

Componente	**Descripción técnica**	**Objetivo / Beneficio**
Compartición de memoria	Las áreas pueden compartirse entre tareas mediante estructuras de referencia y un *pagemap* común.	Comunicación eficiente entre procesos sin duplicar memoria.
Page faults	Los fallos de página se resuelven bajo demanda según el backend del área (anónima, ELF o física).	Carga diferida de memoria y uso eficiente de recursos.
TLB y coherencia	El sistema mantiene coherencia entre TLB y tablas de páginas mediante invalidaciones y *TLB shootdown* en sistemas SMP.	Garantizar consistencia y corrección en ejecución multiprocesador.

4.2.5 Sistema de archivos

El sistema operativo HelenOS tiene soporte de sistemas de archivos basado en un Sistema de Archivos Virtual (VFS), que actúa como una capa de abstracción entre las aplicaciones y los distintos sistemas de archivos. El VFS se implementa como un servidor central encargado de unificar el acceso a los dispositivos de almacenamiento. Cada sistema de archivos se ejecuta como un servicio independiente en espacio de usuario y registra sus capacidades en el VFS. El VFS proporciona una interfaz común de operaciones y ofrece compatibilidad con POSIX mediante una capa de adaptación. Su diseño se divide en un frontend, que gestiona solicitudes simples, y un backend, que delega las operaciones al servidor de archivos correspondiente. (Zárevúcky, Jiří,

2012; Cimerman, Miroslav, 2025).

La Tabla 4.3 detalla los componentes principales del subsistema de archivos de HelenOS, incluyendo el servidor VFS central, los servidores de sistemas de archivos específicos y los componentes de soporte que permiten la gestión unificada del almacenamiento en un entorno multiservidor (Zárevúcky, Jiří, 2012; HelenOS Project, 2006).

Tabla 4.3: Componentes principales del subsistema de archivos en HelenOS

Componente	**Ubicación / Tipo**	**Función principal**
VFS (Virtual File System)	Servidor central en espacio de usuario	Proporciona una interfaz unificada de acceso a archivos, mantiene el árbol global de directorios y coordina las operaciones entre aplicaciones y servidores de sistemas de archivos
Servidores de sistemas de archivos	Servidores independientes en espacio de usuario	Implementan la lógica específica de cada sistema de archivos (FAT, ext2, tmpfs, etc.) y gestionan estructuras y metadatos del almacenamiento

Tabla 4.3 – continuación

Componente	**Ubicación / Tipo**	**Función principal**
Frontend VFS	Módulo interno del servidor VFS	Atiende llamadas básicas de aplicaciones (`open`, `read`, `write`, `close`), administra descriptores y controla archivos abiertos
Backend VFS	Módulo interno del servidor VFS	Traduce operaciones genéricas en solicitudes específicas del sistema de archivos correspondiente y coordina el acceso concurrente
Capa de compatibilidad POSIX	Biblioteca `libc` en espacio de usuario	Mapea llamadas POSIX estándar a la API nativa de HelenOS, permitiendo la portabilidad de aplicaciones
Controladores de dispositivos de bloque	Servidores de dispositivos en espacio de usuario	Gestionan el acceso a dispositivos de almacenamiento y proporcionan una interfaz de bloques al VFS
Caché de bloques	Componente del servidor VFS	Optimiza el rendimiento almacenando bloques frecuentes en memoria y reduciendo accesos a disco

Tabla 4.3 – continuación

Componente	**Ubicación / Tipo**	**Función principal**
Registro de montajes	Estructura interna del VFS	Mantiene información de volúmenes montados, puntos de montaje y servidores responsables

Este diseño modular permite que cada sistema de archivos funcione como un proceso independiente, mejorando la estabilidad del sistema ya que un fallo en un servidor de archivos no afecta al VFS ni a otros sistemas de archivos montados.

4.2.6 Interfaz de usuario

HelenOS proporciona una interfaz de usuario básica basada en texto, sin soporte gráfico integrado en el núcleo. La interacción con el sistema se realiza mediante un shell en espacio de usuario, denominado Bdsh, encargado de interpretar comandos, ejecutar aplicaciones y gestionar la entrada y salida estándar. Dado su enfoque como sistema operativo de investigación, el entorno de usuario es minimalista y se limita a una interfaz de línea de comandos (CLI). Las aplicaciones se comunican con los servidores del sistema a través del API de HelenOS, actuando el shell como un cliente de dichos servicios sin requerir soporte especial del kernel. Segun (HelenOS Project, 2026).

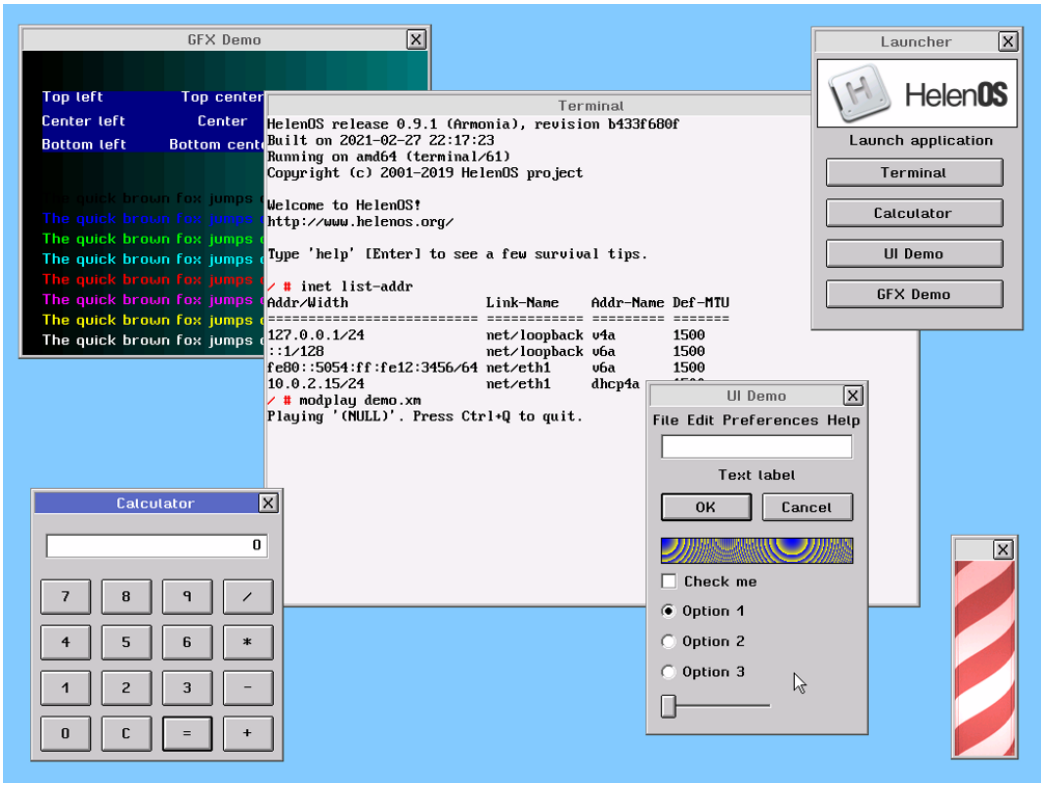


Figura 4.3: Interfaz de usuario de HelenOS (fuente: (HelenOS Project, 2026)).

Internamente está implementado y organizado de la siguiente manera:

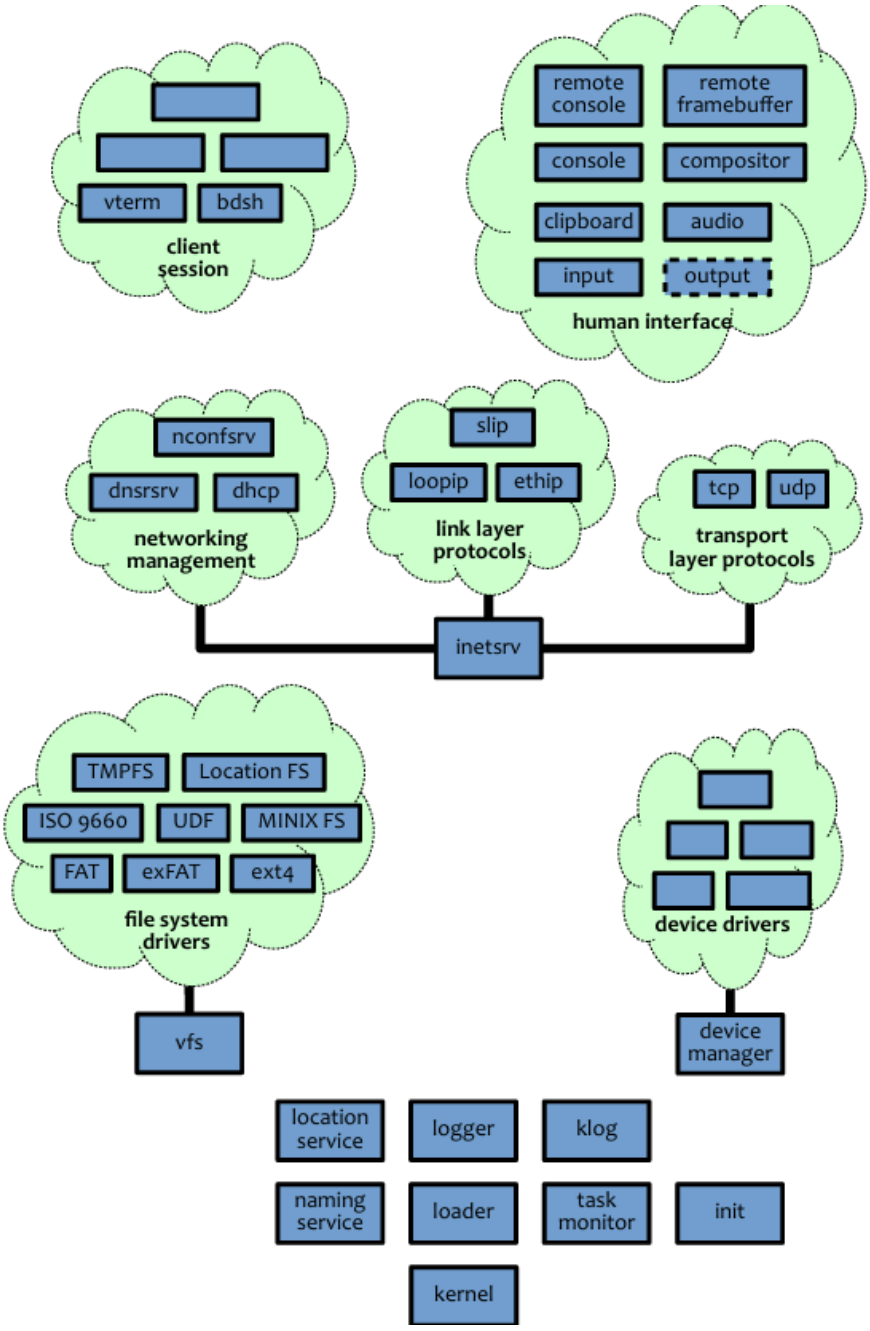


Figura 4.4: Espacio lógico de usuario en HelenOS (fuente: (Děcký, 2015)).

4.2.7 Políticas de planificación y manejo de recursos

HelenOS emplea un planificador preventivo con retroalimentación de prioridad, compatible con sistemas SMP y diseñado para ser altamente portable. Actualmente soporta múltiples arquitecturas de hardware, incluyendo x86, x86-64, IA64, SPARC, PowerPC, ARM y MIPS. Aunque no está orientado al uso general debido a la falta de aplicaciones de usuario, ya cuenta con subsistemas esenciales como sistemas de archivos y redes TCP/IP. El planificador utiliza múltiples colas de ejecución por procesador y una política round-robin con prioridades, incorporando migración de hilos para balancear la carga. El diseño del sistema sigue el principio de separación entre mecanismos y políticas, delegando estas últimas al espacio de usuario. (Děcký, 2010; HelenOS Project, 2006).

Estrategias de planificación y gestión de recursos

La Tabla 4.4 presenta las principales políticas y mecanismos que emplea HelenOS para gestionar la planificación de hilos, la distribución de carga en sistemas multiprocesador y el manejo eficiente de recursos del sistema, siguiendo el principio de separación entre mecanismos (implementados en el kernel) y políticas (definidas en espacio de usuario) (Děcký, 2010; HelenOS Project, 2006).

Tabla 4.4: Políticas de planificación y manejo de recursos en HelenOS

Política / Mecanismo	**Descripción técnica (resumen)**	**Objetivo / Beneficios**
Planificación Round–Robin con prioridades	Cada CPU mantiene colas por nivel de prioridad; los hilos rotan con quantum fijo y se atienden primero las colas de mayor prioridad.	Reparto justo de CPU y respeto de prioridades, evitando inanición mediante envejecimiento.
Afinidad de CPU y migración de hilos	El planificador favorece la ejecución en la CPU previa para aprovechar caché; hilos del kernel migran hilos si detectan desbalance de carga.	Mejor rendimiento por localidad de caché y balance eficiente en sistemas SMP.
Planificación preventiva	El kernel interrumpe hilos mediante temporizador, sin depender de cesión voluntaria de CPU.	Evitar monopolio del procesador y garantizar buena respuesta interactiva.
Retroalimentación de prioridad	La prioridad se ajusta dinámicamente según uso de CPU e I/O del hilo.	Equilibrar throughput y tiempo de respuesta, favoreciendo tareas interactivas.

Tabla 4.4 – continuación

Política / Mecanismo	**Descripción técnica (resumen)**	**Objetivo / Beneficios**
Separación mecanismo–política	El kernel implementa mecanismos genéricos; las políticas se definen en servidores de usuario.	Flexibilidad y experimentación sin modificar el kernel.
Quantum de tiempo adaptativo	La duración del quantum varía según tipo de hilo y carga del sistema.	Reducir overhead de cambio de contexto y mejorar interactividad.
Gestión de recursos por servidor	La gestión de memoria, archivos e IPC se delega a servidores de usuario.	Aislamiento, modularidad y tolerancia a fallos.
Soporte multiprocesador (SMP)	El planificador usa estructuras con bloqueo fino para múltiples CPUs.	Escalabilidad y máximo paralelismo en hardware multicore.

Esta arquitectura flexible permite que HelenOS se adapte a diferentes cargas de trabajo y objetivos de rendimiento sin requerir recompilación del kernel, manteniendo la simplicidad conceptual del microkernel.

HelenOS también posee una manera para gestionar los drivers en forma de árbol tal como se muestra en la siguiente figura:

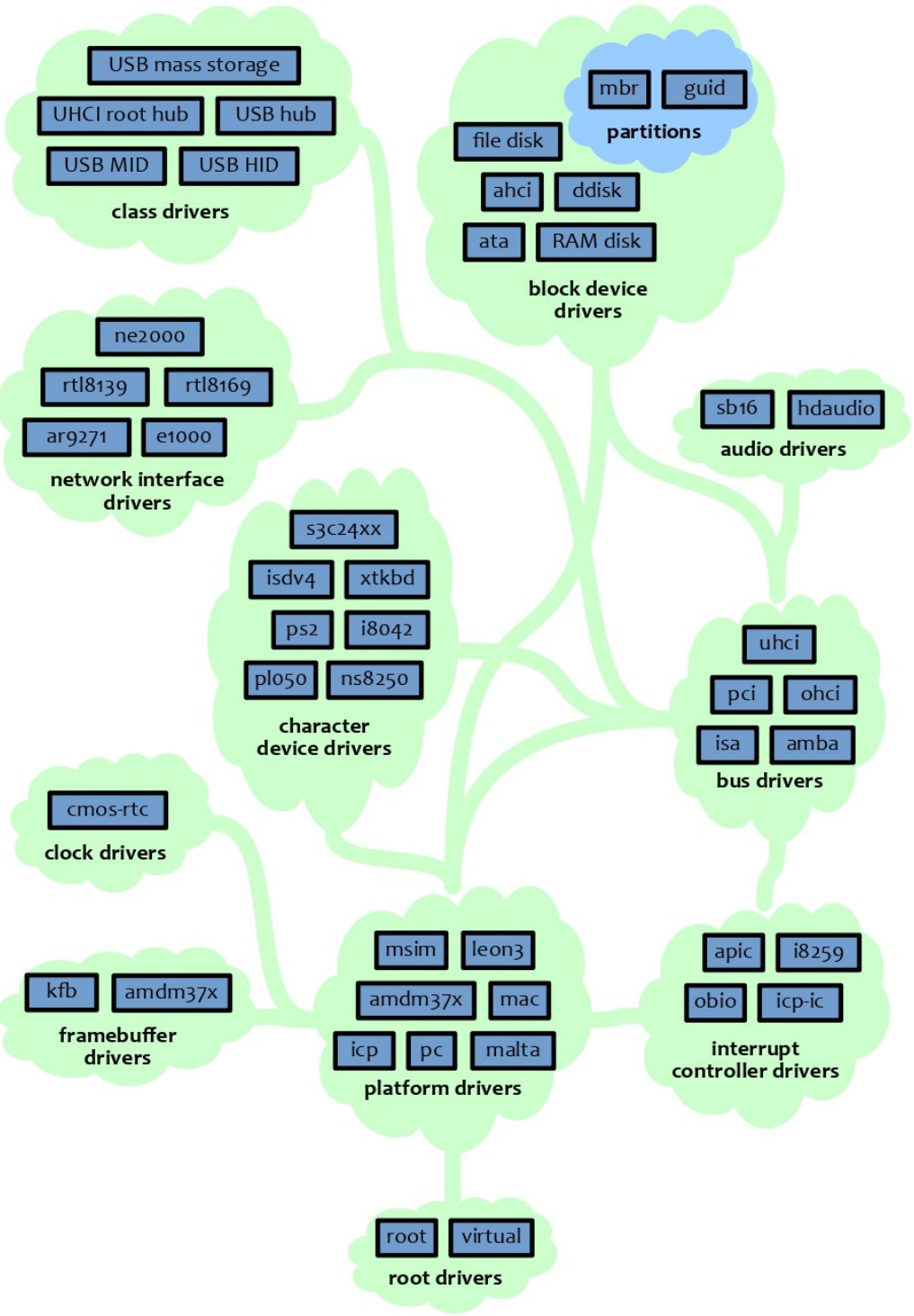


Figura 4.5: Arbol de drivers de HelenOS (fuente: (Děcký, 2015)).

4.2.8 Flujo de ejecución básico

El flujo de ejecución de HelenOS inicia con el bootloader, que carga el microkernel en memoria y transfiere el control. El kernel inicializa el hardware esencial y habilita la planificación, la gestión de memoria y el IPC. Luego se lanzan los servidores fundamentales en espacio de usuario, como controladores de dispositivos, VFS y servicios de nombres. Finalmente, se inicia el shell Bdsh, y el sistema opera mediante múltiples tareas independientes que se comunican de forma concurrente a través del IPC del microkernel.(HelenOS Project, 2006)

Capítulo 5

Herramientas y entorno de desarrollo

HelenOS es un sistema operativo académico multiescala ("microkernel + servidores de usuario") desarrollado en C. Prácticamente todo el código nativo del sistema (núcleo y servidores) está escrito en C, que es el único lenguaje con un *runtime* nativo completo. Algunos componentes muy bajo nivel (por ejemplo, el arranque o manejadores de interrupción) se implementan en ensamblador específico de cada CPU para manejar detalles de hardware. según (Jindrák, Jaroslav, 2022) pag. 3.

5.1 Lenguaje(s) de programación alto y bajo nivel

No existe aún un soporte nativo oficial para C++ o lenguajes de alto nivel: actualmente C++ y Python sólo pueden usarse vía puertos experimentales (Jindrák, Jaroslav, 2022).

Como indica la documentación, HelenOS "puede ejecutar casi exclusivamente pro-

gramas en C... tiene soporte para C++ y Python, pero con limitaciones significativas, por lo que estos lenguajes no se usan ampliamente". Adicionalmente, se han emprendido proyectos de investigación para portar lenguajes modernos; por ejemplo, recientes trabajos buscan habilitar Rust en HelenOS (Volf, Matěj, 2025).

Pordemos observar Interfaz gráfica de HelenOS (versión 0.11.2) la Figura 5.1.

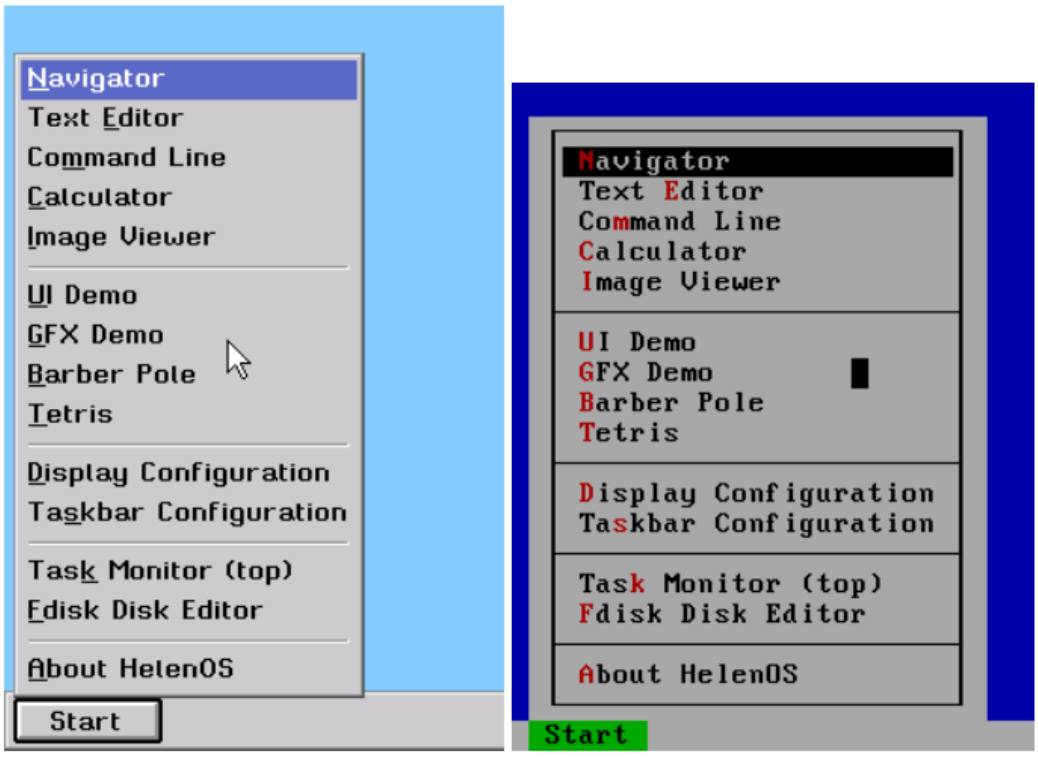


Figura 5.1: Interfaz gráfica y opciones de usuario. (fuente: (HelenOS Project, 2026)).

HelenOS dispone de línea de comandos y un GUI propio para usuario. Como resume la documentación oficial: "Tenemos línea de comandos y una interfaz gráfica sencilla que permiten manipular archivos, ejecutar aplicaciones y montar sistemas de ficheros... se puede jugar al Tetris o editar archivos de texto... También tenemos

redes y se ha portado software de desarrollo (GCC, binutils, Python, pcc)" (HelenOS Project, 2026). En la figura se ve una sesión típica en HelenOS con varias aplicaciones gráficas. En suma, el entorno de usuario de HelenOS incluye utilidades básicas (editor de texto, gestor de archivos, demos gráficas, etc.) soportadas directamente por el sistema, pero el núcleo del OS y sus servidores están escritos casi exclusivamente en C (con ayuda puntual de ensamblador).

5.2 Compilador cruzado

Para compilar HelenOS se requiere un *toolchain* cruzado específico. El proyecto provee un script (`tools/toolchain.sh`) que genera un compilador cross-GCC para cada arquitectura objetivo (amd64, ia32, arm, etc.) ((Děcký, 2015)). No se puede usar el compilador nativo del host para construir HelenOS: la documentación advierte claramente que el compilador del sistema produce binarios incompatibles y que sólo se ha probado HelenOS con la versión de GCC instalada por el script ((Děcký, 2010)). El compilador por defecto de HelenOS es GCC ; existe parche `-helenos-` para hacer el compilador "nativo" de HelenOS. Alternativamente, HelenOS puede construirse con Clang en arquitecturas comunes (ia32, amd64), aunque Clang no cubre todas las plataformas soportadas. el flujo de desarrollo típico es usar GCC (o Clang en X86) como *cross-compiler*, generado por el script oficial, para compilar bibliotecas y aplicaciones HelenOS.

5.3 Emulador

Las pruebas y demostraciones de HelenOS se realizan sobre todo en máquinas virtuales/emuladores. El Emulador recomendado y más usado es QEMU: existe una página dedicada ("Running HelenOS in QEMU") que explica cómo lanzar imágenes de HelenOS en QEMU. El proyecto incluye un script de envoltura (`tools/ew.py`) que inicia automáticamente QEMU con las opciones adecuadas para la configuración elegida. La documentación de HelenOS menciona explícitamente "RunningInVirtualBox"con instrucciones para VirtualBox, y en el repositorio se proporciona incluso un ejemplo de configuración de VMware (archivo `.vmx`) para arrancar HelenOS en VMware Workstation. (HelenOS Project, 2026).

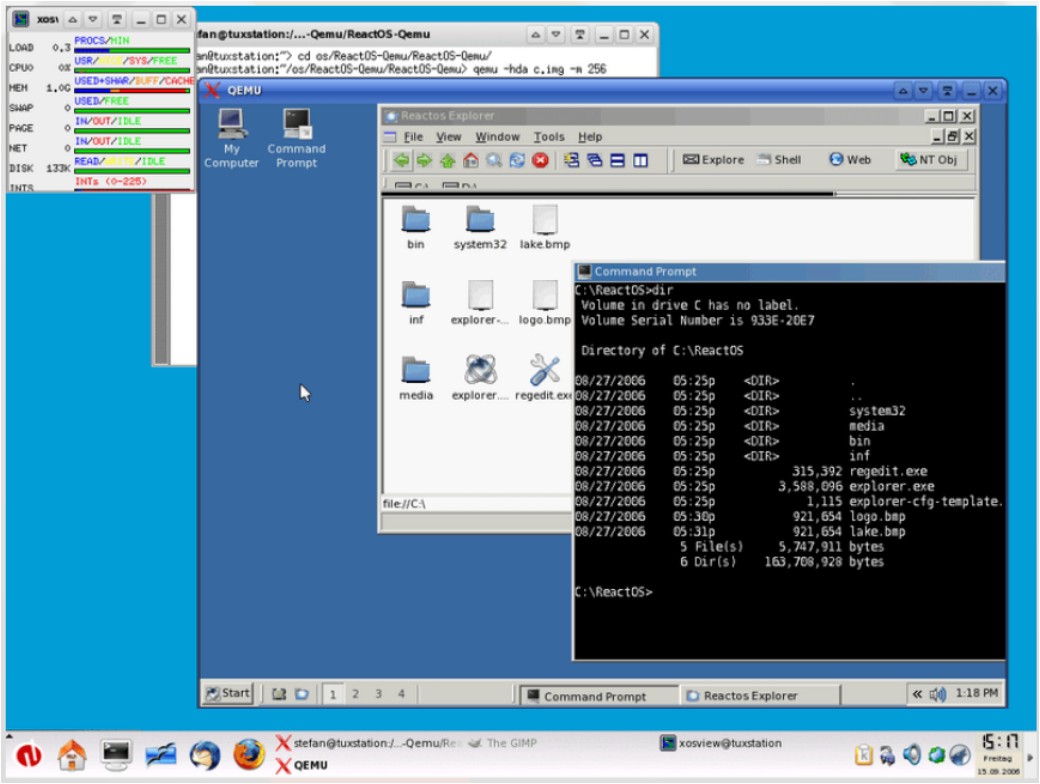


Figura 5.2: QEMU es un emulador y virtualizador de código abierto capaz de realizar emulación de sistema completo en múltiples arquitecturas (fuente: (HelenOS Project, 2026)).

5.4 Control de versiones (Git)

HelenOS gestiona su código fuente con Git. Los repositorios oficiales están alojados en GitHub: el repositorio principal ((HelenOS Project, 2026)) contiene el OS base, y existen repositorios adicionales ("harbours" para software portado y "ci" para integración continua). En sus FAQ oficiales se indica que "el código fuente de HelenOS actualmente se gestiona con Git... los desarrolladores son alentados a alojar sus ramas en GitHub (github.com/HelenOS)". Antes se usaba Bazaar, pero hoy todo se ha

migrado a Git. Como buenas prácticas, el equipo mantiene ramas bien identificadas (por versión/milestone) y usa revisiones por *pull requests* en GitHub.

5.5 Editor o entorno de desarrollo

La documentación oficial de HelenOS no prescribe un editor de código o IDE específico para desarrollar el sistema. En la práctica, los desarrolladores utilizan los editores y entornos comunes de la comunidad (C/C++) que prefieran. Muchos usan editores de texto avanzados (por ejemplo, **Vim o Emacs**) o entornos gráficos como **Visual Studio Code** con extensiones de C/C++, dado que HelenOS es esencialmente software en C. HelenOS incluye también un editor básico propio (comando `edit` en la consola), pero para escribir el código fuente generalmente se prefiere un editor externo con resaltado de sintaxis y soporte de *debugging*. En ausencia de una recomendación oficial, cada desarrollador escoge su entorno de desarrollo habitual (p.ej. VS Code, CLion, Vim/Neovim, etc.) para editar el código de HelenOS.

Capítulo 6

Planificación de implementación

6.1 Cronograma tentativo por componentes.

Tabla 6.1: Cronograma de Implementación (24 Nov 2025 – 02 Ene 2026)

Semana	**Fechas**	**Actividades Principales**	**Descripción Ampliada**	**Componentes Involucrados**
1	24 Nov – 30 Nov	Configuración del entorno de desarrollo y compilación inicial	Se prepara el entorno completo de desarrollo de HelenOS. Esto incluye la instalación del compilador cruzado (toolchain), la descarga del código fuente oficial desde el repositorio, la verificación del sistema de construcción (Meson/Ninja) y la ejecución de la primera compilación completa. Se valida la ejecución del sistema en QEMU para asegurar que el pipeline de desarrollo esté correctamente configurado.	Toolchain; Build System; QEMU

Continúa en la siguiente página

Semana	Fechas	Actividades Principales	Descripción Ampliada	Componentes Involucrados
2	01 Dic – 07 Dic	Análisis de arquitectura y diseño técnico	Se realiza un análisis profundo de la arquitectura de HelenOS, estudiando su microkernel, el mecanismo IPC (Inter-Process Communication), la administración de tareas y el VFS modular. Se identifican los módulos clave a estudiar y se elabora el diseño técnico inicial, incluyendo diagramas de interacción y estructuras de datos que servirán como guía para las siguientes fases.	Kernel; IPC; VFS; Documentación

Continúa en la siguiente página

Semana	Fechas	Actividades Principales	Descripción Ampliada	Componentes Involucrados
3	08 Dic – 14 Dic	Estudio del Kernel y gestión de procesos	Se estudian las funciones internas del kernel relacionadas con la administración de procesos, incluyendo el scheduler (planificador de tareas). Se analizan las estructuras de control de procesos (PCB), los estados de los procesos y los mecanismos de cambio de contexto. Se documentan los hallazgos y se realizan pruebas de comprensión mediante la modificación de parámetros básicos.	Kernel; Scheduler; Gestión de Procesos

Continúa en la siguiente página

Semana	Fechas	Actividades Principales	Descripción Ampliada	Componentes Involucrados
4	15 Dic – 21 Dic	Estudio del VFS y sistema de archivos	Se analiza el Sistema de Archivos Virtual (VFS) de HelenOS, comprendiendo las operaciones esenciales como lectura, escritura, apertura y cierre de archivos. Se estudia la integración entre el kernel y el VFS, así como los comandos disponibles en la shell (Bdsh) para interactuar con el sistema de archivos. Se documenta la arquitectura del VFS y sus interfaces.	VFS; Kernel; Shell (Bdsh)

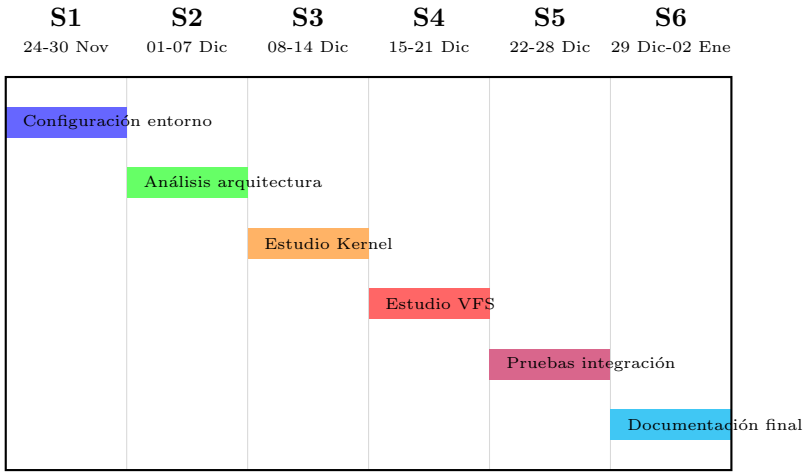
Continúa en la siguiente página

Semana	Fechas	Actividades Principales	Descripción Ampliada	Componentes Involucrados
5	22 Dic – 28 Dic	Pruebas de integración y validación del sistema	Se ejecutan pruebas integrales del sistema operativo en el emulador QEMU, evaluando la interoperabilidad entre componentes, el manejo de recursos, la estabilidad del scheduler y la consistencia del sistema de archivos. Se documentan los resultados de las pruebas y se identifican posibles mejoras o áreas de interés para trabajo futuro.	Todos los componentes; QEMU

Continúa en la siguiente página

Semana	**Fechas**	**Actividades Principales**	**Descripción Ampliada**	**Componentes Involucrados**
6	29 Dic – 02 Ene	Documentación final y conclusiones	Se elabora la documentación técnica final del proyecto, incluyendo diagramas actualizados, decisiones arquitectónicas analizadas, resultados de pruebas y conclusiones del estudio. Se prepara la presentación del trabajo y se revisan todos los entregables para asegurar coherencia y completitud del documento final.	Documentación; Presentación

6.2 Diagrama de Gantt



6.3 Estrategia de pruebas y validación.

1. Pruebas unitarias

- Funciones pequeñas del kernel (manejo de listas, asignación de estructuras, validaciones internas).
- Validación manual y automática mediante logs y assert estructurado.

2. Pruebas de integración

- Validación del pipeline completo: arranque del kernel → inicialización → scheduler → VFS → CLI.
- Empleo de scripts en QEMU para generar escenarios repetibles.

3. Pruebas de regresión

- Comparación entre versiones para asegurar que cambios recientes **no rompen el arranque** ni la CLI.

4. Pruebas de experiencia de usuario

- Comandos del shell, manejo de errores, mensajes coherentes, terminación de procesos inválidos.

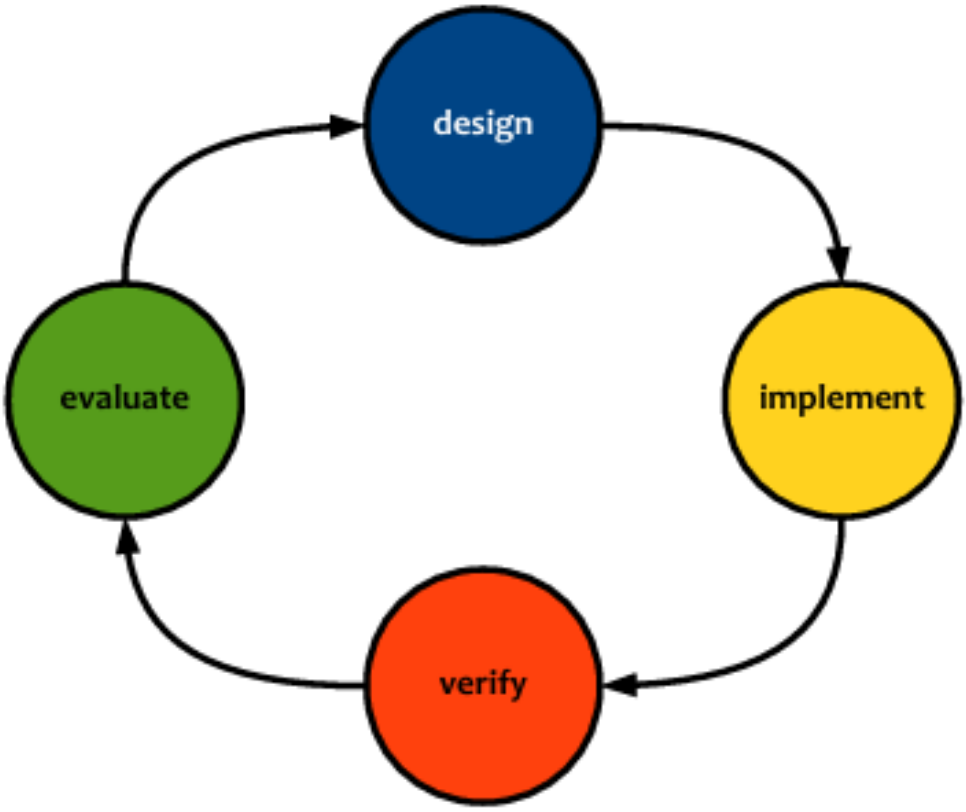


Figura 6.1: Ciclo de desarrollo de HelenOS (fuente: (Děcký, 2015)).

6.4 Posibles riesgos y cómo mitigarlos.

Tabla 6.2: Tabla de riesgos ampliada

Riesgo	**Descripción técnica**	**Impacto**	**Probabilidad**	**Mitigación**	**Indicadores tempranos**
Fallas al construir el toolchain	GCC o Binutils no compilan correctamente para la arquitectura objetivo	Alto	Media	Usar script oficial, verificar dependencias, versiones compatibles	Errores de linking o binarios incompletos
Kernel panic o page faults	Punteros nulos, errores en manejo de interrupciones o memoria	Alto	Alta	Validar punteros, usar logs del kernel en cada módulo	Reinicios inesperados, caída antes del scheduler

Continúa en la siguiente página

Riesgo	**Descripción técnica**	**Impacto**	**Probabilidad**	**Mitigación**	**Indicadores tempranos**
Integración inconsistente entre módulos	Cambios en kernel que afectan procesos o VFS	Alto	Media	Commits pequeños, pruebas continuas, interfaces claras	Errores en syscalls o creación de procesos
Falta de documentación interna	Dificultad para comprender estructuras internas del kernel	Medio	Alta	Revisar código fuente, analizar diagramas, revisar tesis y reportes	Bloqueos del equipo en análisis
Baja performance o deadlocks	Scheduler inestable o locks mal implementados	Alto	Media	Scheduler simple, no agregar complejidad innecesaria	Procesos que no terminan o se congelan

Continúa en la siguiente página

Riesgo	**Descripción técnica**	**Impacto**	**Probabilidad**	**Mitigación**	**Indicadores tempranos**
Escasez de tiempo del equipo	Tareas que se extienden más de lo esperado	Medio	Media	Dividir roles, Scrum semanal, priorizar mínimo viable	Retraso desde semana 2 en adelante
VFS incompleto o inconsistente	Lectura/escritura fallida o tabla de inodos incorrecta	Medio	Media	Implementación incremental, validar caso mínimo	Archivos que no montan, errores en CLI
Problemas de compatibilidad en QEMU	Emulación inconsistente o flags incorrectos	Bajo	Media	Usar parámetros probados, logs seriales	QEMU se congela o no muestra consola
Sobrecarga en el kernel educativo	Añadir funciones innecesarias o fuera del alcance	Medio	Baja	Control estricto del alcance (scope)	Demoras y complejidad excesiva

Referencias

Cimerman, Miroslav (2025). Software RAID for HelenOS. Tesis de Grado (Bakalářská práce), Charles University (Univerzita Karlova), Faculty of Mathematics and Physics, Praga, República Checa. Supervisor: Mgr. Vojtěch Horký, Ph.D. Departamento de Sistemas Distribuidos y Confiables. Programa: Computer Science. Accedido el 4 de enero de 2026.

Děcký, M. (2010). A road to a formally verified general-purpose operating system. In *Proceedings of the International Symposium on Architecting Critical Systems (ISARCS '10), Lecture Notes in Computer Science, vol. 6150.* Recuperado el 22 Octubre 2025.

Děcký, M. (2015). *Application of Software Components in Operating System Design.* PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics.

HelenOS Project (2006). *HelenOS 0.2.0 Design Documentation.* Technical report, Charles University (Univerzita Karlova), Faculty of Mathematics and Physics. Documentación oficial de diseño y principios del micronúcleo. Accedido el 3 de enero de 2026.

HelenOS project (2026). HelenOS: The multiserver microkernel-based operating sys-

tem. Sitio web oficial. Accedido el 3 de enero de 2026. Fuente de la imagen de arquitectura y diseño del sistema.

HelenOS Project (2026). HelenOS Wiki: Tutorial. Wiki oficial del proyecto HelenOS. Instrucciones de ejecución y despliegue del sistema en arquitectura x86. Accedido el 3 de enero de 2026.

Jermář, J. (2025a). Helenos — continuous integration tools and automated tests (github repository). https://github.com/HelenOS/ci. Repositorio accedido el 22 Octubre 2025.

Jermář, J. (2025b). Helenos — source code (github repository). https://github.com/HelenOS/helenos. Repositorio accedido el 22 Octubre 2025.

Jermář, Jakub and Děcký, Martin and Cejpek, Josef and Mejdrech, Lukáš and others (2006). *HelenOS - Software Project Documentation*. Technical report, Charles University (Univerzita Karlova), Faculty of Mathematics and Physics. Documentación detallada del proyecto de software. Accedido el 3 de enero de 2026.

Jindrák, Jaroslav (2022). C++ Runtime for HelenOS. Tesis de Maestría (Diplomová práce), Charles University (Univerzita Karlova), Faculty of Mathematics and Physics, Praga, República Checa. Supervisor: Mgr. Martin Děcký, Ph.D. Departamento de Sistemas Distribuidos y Confiables. Programa: Computer Science. Rama: Software Systems. Accedido el 4 de enero de 2026.

Korop, N. (2025). Captura de paquetes para helenos. Tesis de grado, Universidad Charles (Univerzita Karlova), Facultad de Matemáticas y Física. Aceptada el 19 de junio de 2025. Accedido el 3 de enero de 2026.

SCRUMstudy (2022). Guía de los fundamentos de scrum (guía del sbok®) — cuarta edición (spanish). https://scrumstudy.pe/wp-content/uploads/2025/01/SCRUMstudy-SBOK-Guide-4th-Edition-Spanish.pdf. Guía accedida el 22 Octubre 2025; publicada por SCRUMstudy™, cuatro edición del SBOK® Guide, disponible en español.

Volf, Matěj (2025). Rust for HelenOS. Tesis de Grado (Bakalářská práce), Charles University (Univerzita Karlova), Faculty of Mathematics and Physics, Praga, República Checa. Director de tesis: Mgr. Vojtěch Horký, Ph.D. Departamento de Sistemas Distribuidos y Confiables. Accedido el 3 de enero de 2026.

Zárevúcky, Jiří (2012). Improved VFS design for HelenOS. Tesis de Grado (Bakalářská práce), Masaryk University (Masarykova univerzita), Faculty of Informatics, Brno, República Checa. Accedido el 4 de enero de 2026. Este trabajo rediseña el servidor de archivos virtual para optimizar la comunicación IPC en HelenOS.

Anexos

Anexo A: Comandos básicos de HelenOS

La siguiente tabla presenta los comandos más utilizados en el shell Bdsh de HelenOS, que permiten la interacción básica con el sistema operativo:

Tabla 6.3: Comandos básicos del shell Bdsh en HelenOS

Comando	**Descripción**
`help`	Muestra la lista de comandos disponibles en el shell.
`ls [directorio]`	Lista el contenido del directorio especificado o del directorio actual.
`cd <directorio>`	Cambia al directorio especificado.
`pwd`	Muestra la ruta del directorio de trabajo actual.
`cat <archivo>`	Muestra el contenido de un archivo de texto.
`cp <origen> <destino>`	Copia un archivo del origen al destino.
`rm <archivo>`	Elimina el archivo especificado.

Comando	Descripción
`mkdir <directorio>`	Crea un nuevo directorio.
`mount <tipo> <dispositivo> <punto>`	Monta un sistema de archivos.
`unmount <punto>`	Desmonta un sistema de archivos.
`tasks`	Muestra las tareas actualmente en ejecución.
`threads`	Muestra los hilos del sistema.
`kill <id>`	Termina la tarea con el identificador especificado.
`klog`	Muestra el registro del kernel.
`exit`	Finaliza la sesión del shell.

Anexo B: Estructura del código fuente de HelenOS

El repositorio de HelenOS sigue una organización jerárquica que refleja su arquitectura modular. A continuación se describe la estructura principal de directorios:

Tabla 6.4: Estructura de directorios del código fuente de HelenOS

Directorio	Contenido
`/kernel`	Código fuente del micronúcleo, incluyendo subsistemas de memoria, planificación, IPC y soporte de arquitecturas.

Directorio	Contenido
`/kernel/arch`	Código específico de cada arquitectura soportada (ia32, amd64, arm32, mips32, etc.).
`/kernel/generic`	Código genérico del kernel independiente de la arquitectura.
`/uspace`	Código de espacio de usuario, incluyendo bibliotecas, servidores y aplicaciones.
`/uspace/lib`	Bibliotecas de espacio de usuario (libc, libposix, libui, etc.).
`/uspace/srv`	Servidores del sistema (vfs, devman, locsrv, ns, etc.).
`/uspace/drv`	Controladores de dispositivos en espacio de usuario.
`/uspace/app`	Aplicaciones de usuario (bdsh, edit, tetris, etc.).
`/boot`	Archivos de arranque y configuración del bootloader.
`/tools`	Scripts y herramientas de desarrollo (toolchain.sh, ew.py, etc.).
`/abi`	Definiciones de la interfaz binaria de aplicaciones.

Anexo C: Proceso de compilación de HelenOS

El proceso de compilación de HelenOS requiere seguir una serie de pasos específicos. A continuación se presenta la secuencia de comandos necesaria para compilar el sistema desde el código fuente:

```
# 1. Clonar el repositorio
git clone https://github.com/HelenOS/helenos.git
```

```
cd helenos

# 2. Instalar dependencias (Ubuntu/Debian)
sudo apt-get install build-essential wget texinfo flex bison \
    libgmp-dev libmpfr-dev libmpc-dev python3 python3-yaml \
    genisoimage xorriso

# 3. Compilar el toolchain cruzado (ejemplo para amd64)
cd tools
./toolchain.sh amd64
cd ..

# 4. Configurar la arquitectura objetivo
make PROFILE=amd64

# 5. Compilar HelenOS
make

# 6. Ejecutar en QEMU
tools/ew.py
```

El script `toolchain.sh` descarga, configura y compila automáticamente las versiones compatibles de GCC y Binutils para la arquitectura seleccionada. Este proceso puede tomar varios minutos dependiendo de la capacidad del sistema.

Anexo D: Configuración de QEMU para HelenOS

Para ejecutar HelenOS en el emulador QEMU, se pueden utilizar las siguientes opciones de configuración:

```
# Ejecución básica para arquitectura amd64
qemu-system-x86_64 -cdrom image.iso -m 256M -enable-kvm

# Ejecución con salida serial para depuración
qemu-system-x86_64 -cdrom image.iso -m 256M \
    -serial stdio -enable-kvm

# Ejecución con red habilitada
qemu-system-x86_64 -cdrom image.iso -m 256M \
    -netdev user,id=net0 -device e1000,netdev=net0

# Ejecución con múltiples CPUs (SMP)
qemu-system-x86_64 -cdrom image.iso -m 512M -smp 4
```

El script `tools/ew.py` incluido en el repositorio de HelenOS automatiza la configuración de QEMU con los parámetros óptimos para cada arquitectura.

Anexo E: Diagrama de comunicación IPC

El mecanismo de comunicación entre procesos (IPC) en HelenOS se basa en el modelo de paso de mensajes mediante "teléfonos" (phones) y "answerboxes". El flujo típico

de una llamada IPC es el siguiente:

1. El cliente obtiene un teléfono (phone) conectado al servidor destino.
2. El cliente envía un mensaje a través del teléfono usando `ipc_call_sync()` o `ipc_call_async()`.
3. El kernel copia el mensaje corto (hasta 5 argumentos de 64 bits) al answerbox del servidor.
4. El servidor recibe el mensaje mediante `ipc_wait_for_call()`.
5. El servidor procesa la solicitud y prepara la respuesta.
6. El servidor envía la respuesta usando `ipc_answer()`.
7. El kernel entrega la respuesta al cliente, desbloqueándolo si la llamada era síncrona.

El siguiente diagrama ejemplifica este flujo de comunicación:

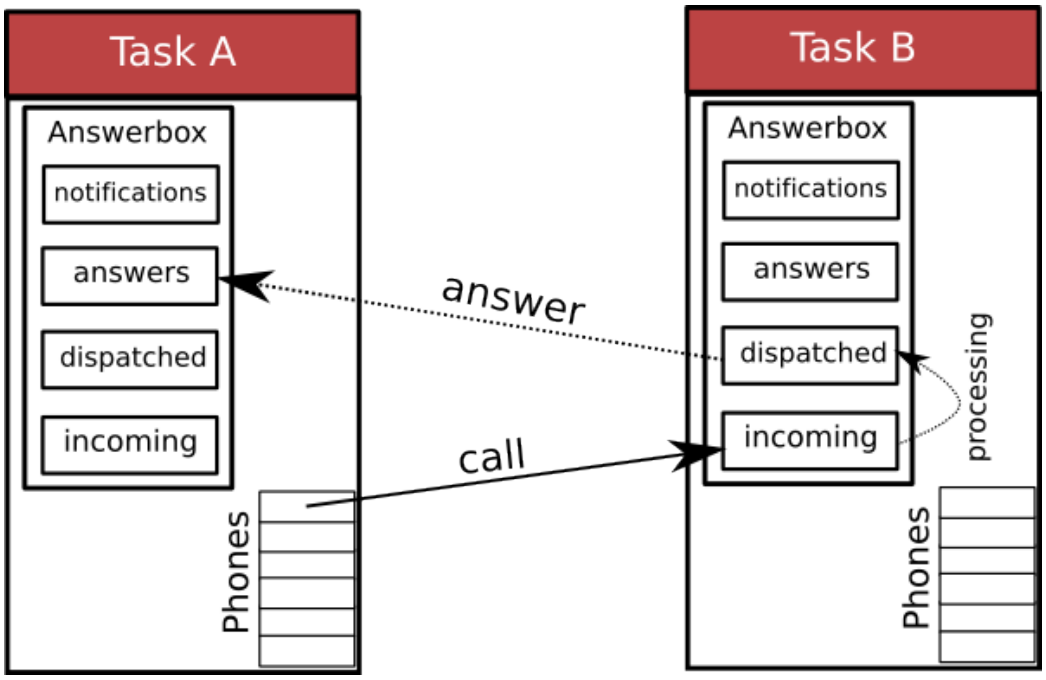


Figura 6.2: Diagrama de comunicación IPC en HelenOS (fuente: (HelenOS Project, 2026)).

Para transferencias de datos mayores, HelenOS utiliza áreas de memoria compartida que se establecen mediante llamadas específicas del IPC, evitando así la copia excesiva de datos entre espacios de direcciones.

Anexo F: Glosario de términos

Tabla 6.5: Glosario de términos técnicos

Término	**Definición**
Microkernel	Arquitectura de sistema operativo donde el núcleo implementa solo funciones esenciales (IPC, planificación básica, gestión de memoria), delegando servicios a procesos de usuario.
Multiservidor	Diseño donde los servicios del sistema operativo se ejecutan como servidores independientes en espacio de usuario.
IPC	Inter-Process Communication. Mecanismo que permite la comunicación entre procesos aislados.
VFS	Virtual File System. Capa de abstracción que unifica el acceso a diferentes sistemas de archivos.
Tarea (Task)	Unidad de recursos en HelenOS que agrupa hilos y define un espacio de direcciones.
Hilo (Thread)	Unidad básica de ejecución en el kernel de HelenOS.
Fibrilla (Fibril)	Hilo cooperativo en espacio de usuario, implementado sobre la API del kernel.
SLAB allocator	Algoritmo de asignación de memoria optimizado para objetos pequeños y frecuentes.
Buddy system	Algoritmo de asignación de memoria física que maneja bloques de tamaño potencia de dos.

Término	Definición
TLB	Translation Lookaside Buffer. Caché de traducciones de direcciones virtuales a físicas.
SMP	Symmetric Multiprocessing. Arquitectura con múltiples procesadores compartiendo memoria.
Teléfono (Phone)	Abstracción de HelenOS para representar un canal de comunicación IPC hacia un servidor.
Cross-compiler	Compilador que genera código para una plataforma diferente a la que se ejecuta.
Bdsh	Brain-dead shell. Shell de línea de comandos por defecto en HelenOS.

Anexo G: Recursos adicionales

Para profundizar en el estudio de HelenOS, se recomiendan los siguientes recursos:

- **Sitio web oficial:** https://www.helenos.org/ – Contiene documentación, noticias y enlaces a recursos del proyecto.
- **Repositorio GitHub:** https://github.com/HelenOS/helenos – Código fuente completo del sistema operativo.
- **Wiki del proyecto:** https://www.helenos.org/wiki/ – Tutoriales, guías de desarrollo y documentación técnica.
- **Documentación de diseño:** https://www.helenos.org/doc/design.pdf –

Documento técnico sobre la arquitectura y diseño del sistema.

- **Tesis relacionadas:** El repositorio de la Universidad Charles de Praga contiene múltiples tesis de grado y maestría sobre diferentes aspectos de HelenOS.
- **Canal IRC:** `#helenos` en `irc.libera.chat` – Canal de comunicación con la comunidad de desarrolladores.

Anexo H: Guía de operación paso a paso

Este manual detalla el flujo de interacción con HelenOS 0.14.1 (Aladar) desde el encendido hasta el apagado del sistema.

Paso 1: Selección en el gestor de arranque

Al iniciar la máquina virtual en QEMU, la primera pantalla que aparece es el menú de GNU GRUB. El usuario debe utilizar las teclas de dirección para resaltar la opción `HelenOS 0.14.1` y presionar la tecla *Enter* para iniciar el proceso de carga del núcleo.

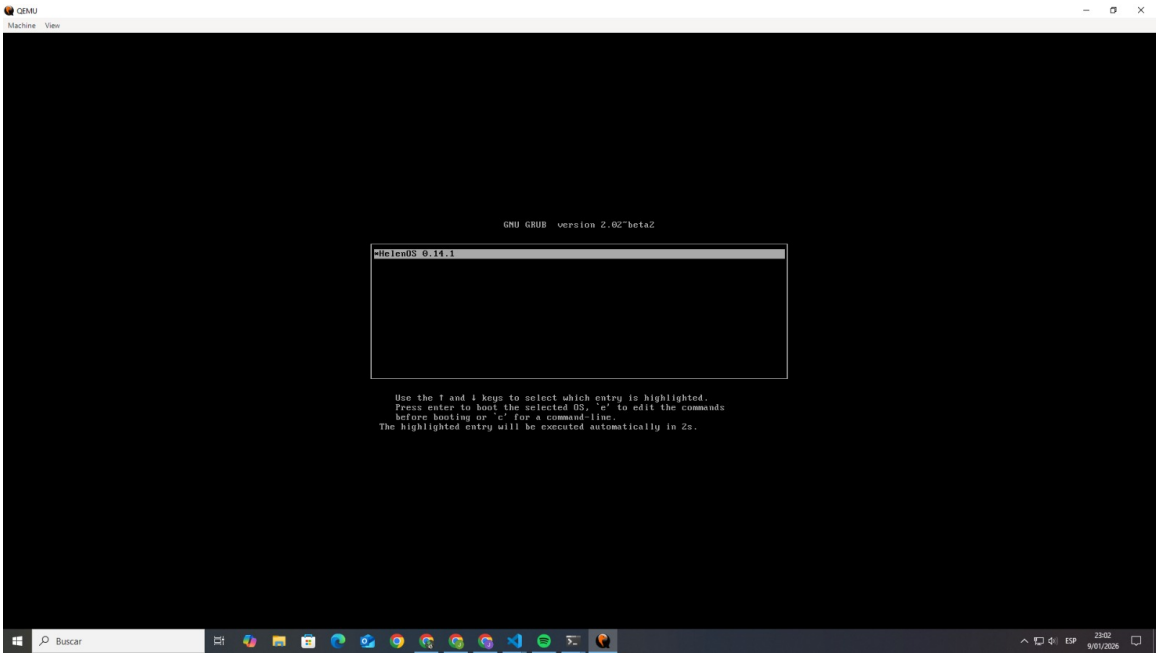


Figura 6.3: Menú de inicio de GRUB para la selección del sistema operativo.

Paso 2: Carga de módulos del sistema

Una vez seleccionado el sistema, se muestra una pantalla de carga de archivos en modo texto. En este punto, el cargador transfiere a la memoria RAM el kernel, los servidores básicos (como el sistema de archivos virtual y el servidor de nombres) y la imagen del disco RAM inicial (`initrd.img`).

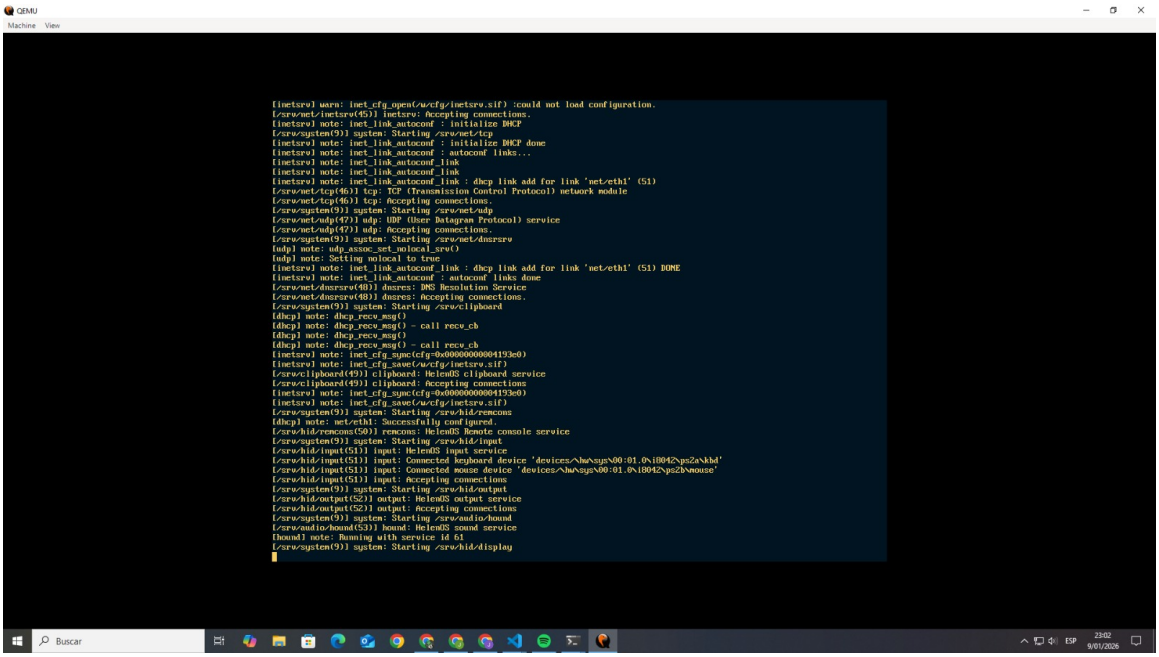


Figura 6.4: Proceso de carga de módulos y componentes esenciales en memoria.

Paso 3: Inicialización y log de servicios

Posteriormente, el sistema comienza a levantar los servicios de hardware y red. El usuario podrá observar en tiempo real cómo se configuran las conexiones de red (DHCP), los dispositivos de entrada (teclado y ratón) y el servidor de video, finalizando con el inicio del servicio de sonido `hound` y el servicio de pantalla.

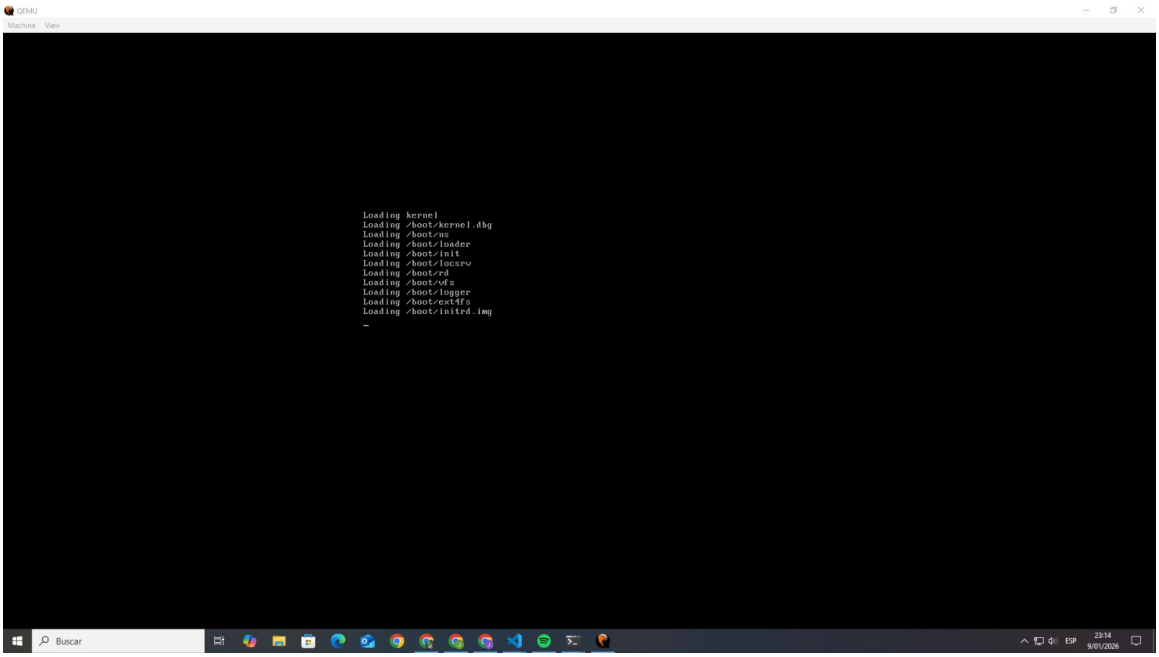


Figura 6.5: Registro de inicialización de servicios de red y hardware.

Paso 4: Ingreso al entorno de escritorio

Tras la carga de servicios, aparece el escritorio gráfico azul. Por defecto, se abre una ventana de Terminal que da la bienvenida al usuario con información sobre la arquitectura (amd64) y la revisión del sistema. En esta ventana se puede escribir el comando `help` para ver consejos de uso.

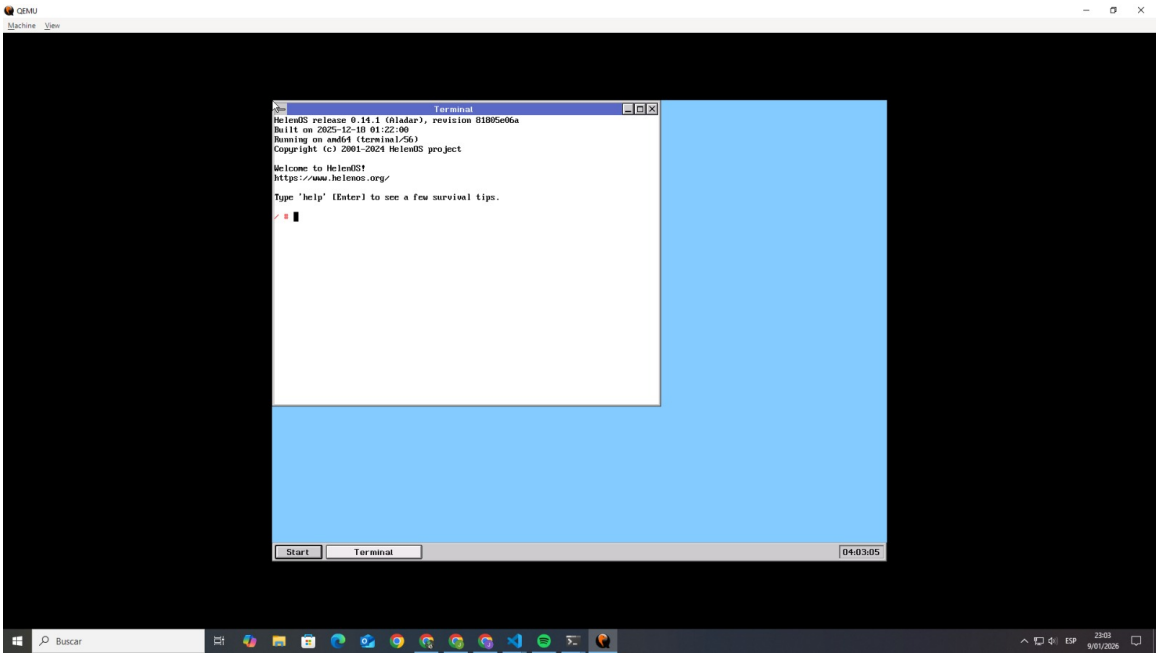


Figura 6.6: Pantalla de bienvenida y terminal inicial en el escritorio.

Paso 5: Monitoreo y herramientas de cálculo

El usuario puede interactuar con múltiples aplicaciones simultáneamente. En la captura se observa el uso del comando `top` en la terminal para monitorear el consumo de recursos (CPU y memoria) y el estado de las tareas. Al mismo tiempo, se puede utilizar la *Calculator* para realizar operaciones matemáticas básicas.

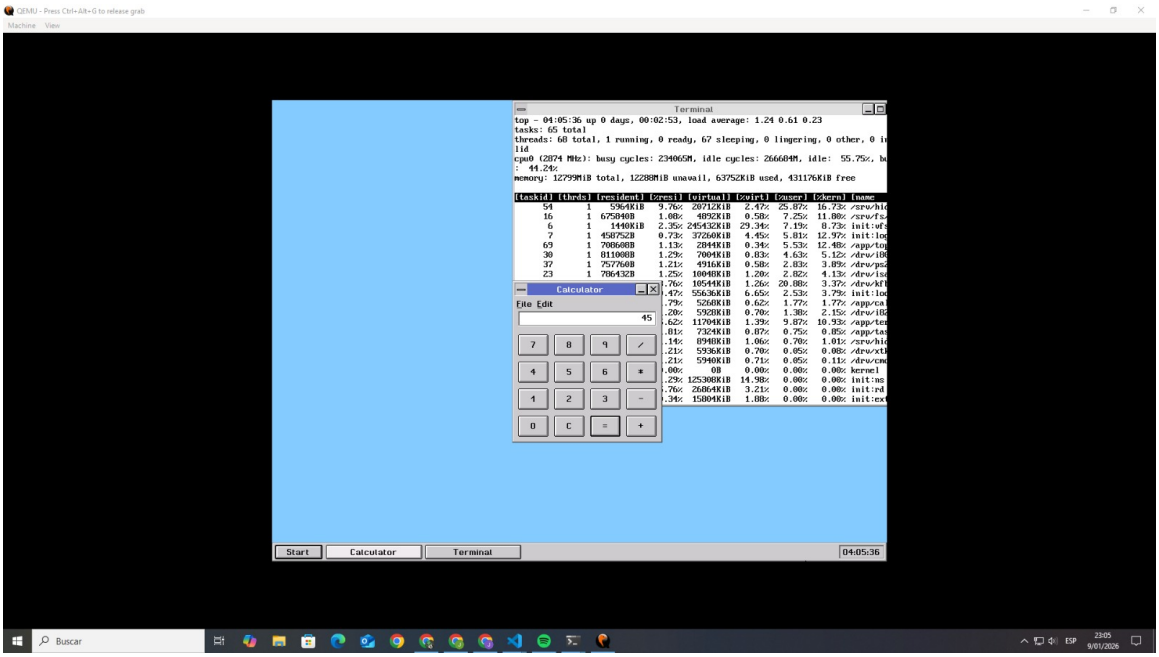


Figura 6.7: Uso simultáneo del monitor de sistema (top) y la calculadora gráfica.

Paso 6: Entretenimiento y juegos

HelenOS incluye aplicaciones de ocio como el juego Tetris, ejecutable dentro de la terminal. Para jugar, el usuario utiliza las teclas `j` (izquierda), `l` (derecha) y `k` (rotar). El sistema permite mantener el monitor de procesos activo en segundo plano mientras se juega.

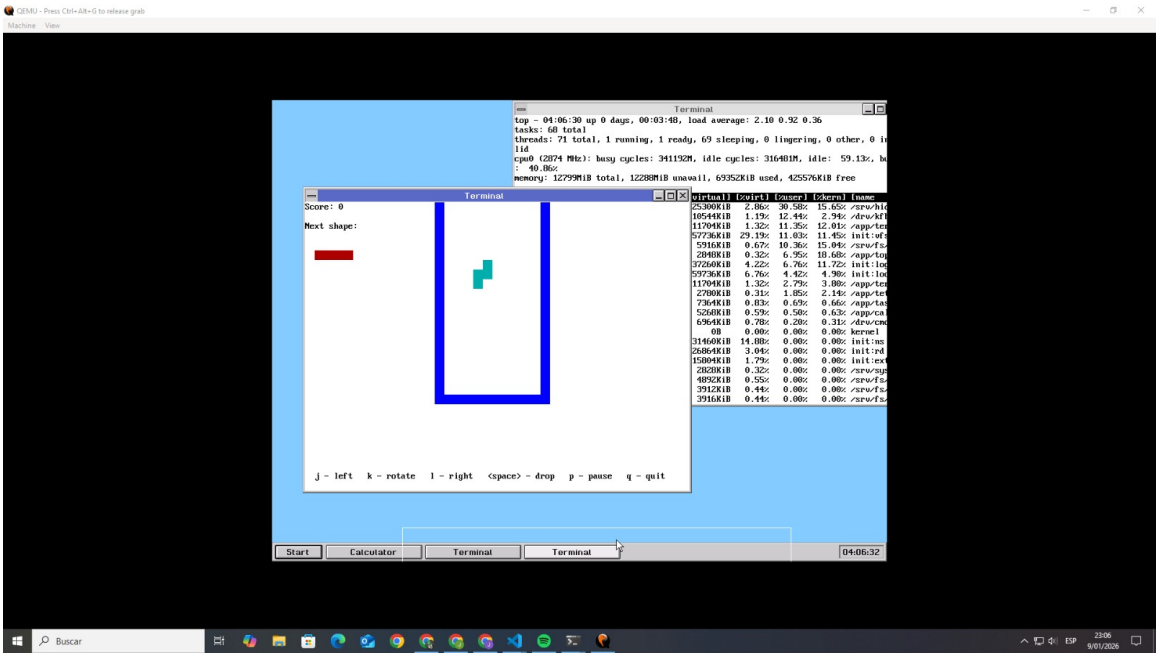


Figura 6.8: Ejecución del juego Tetris en el entorno de consola.

Paso 7: Gestión de ventanas y Barber Pole

La barra de tareas inferior permite alternar entre las diferentes ventanas abiertas. Además de las herramientas de texto, se puede ejecutar la aplicación *Barber Pole*, que muestra una animación gráfica constante. La interfaz soporta el cambio de foco y la organización de múltiples utilidades en pantalla.

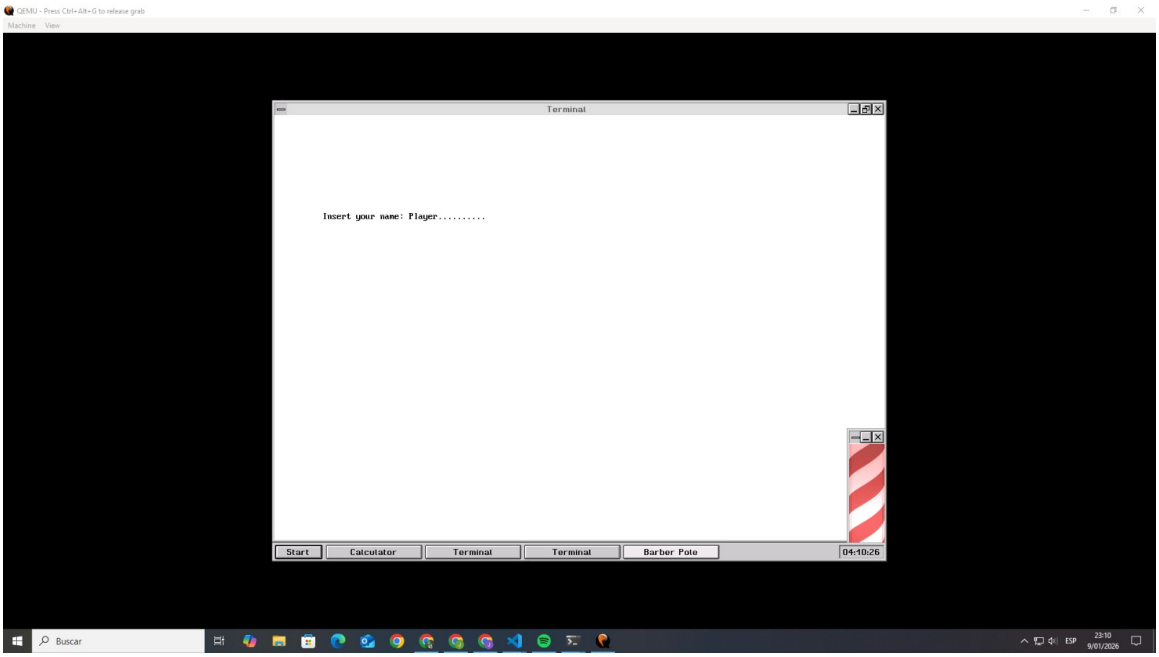


Figura 6.9: Interfaz con múltiples tareas activas y demostración gráfica.

Paso 8: Apagado del sistema

Para finalizar la sesión, el usuario debe hacer clic en el botón *Start* y seleccionar la opción de salida. Se desplegará una ventana de confirmación denominada *Shutdown*, donde se puede elegir entre *Power off* para apagar completamente la máquina virtual o *Restart* para reiniciar el sistema operativo.

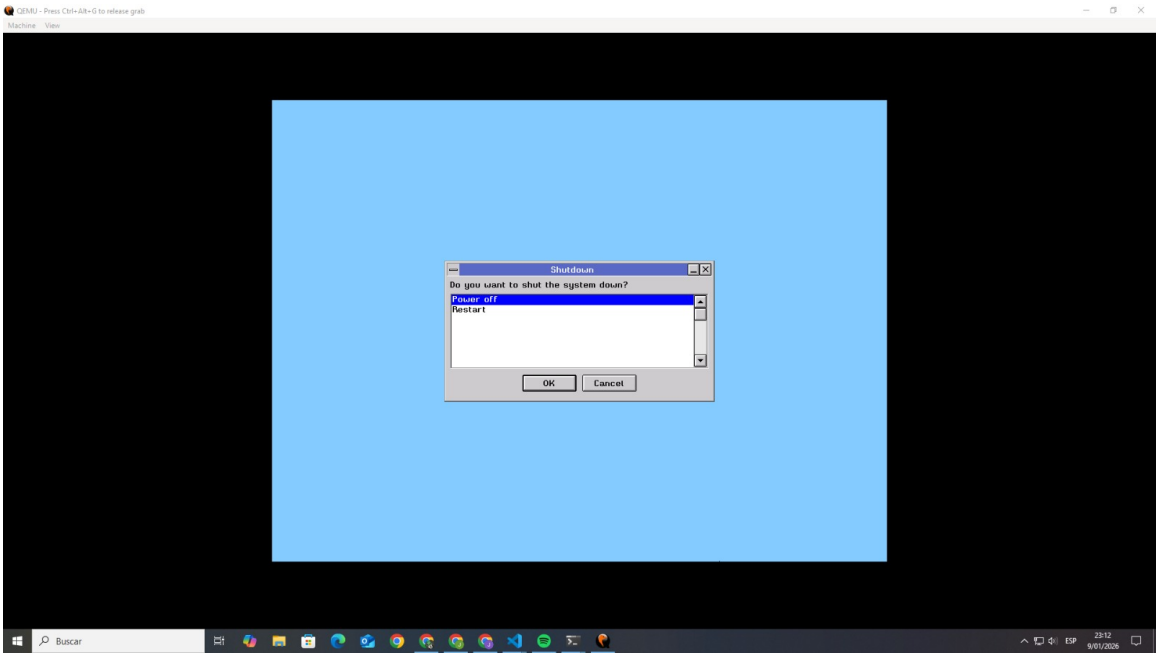


Figura 6.10: Menú de confirmación para el apagado seguro o reinicio.