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

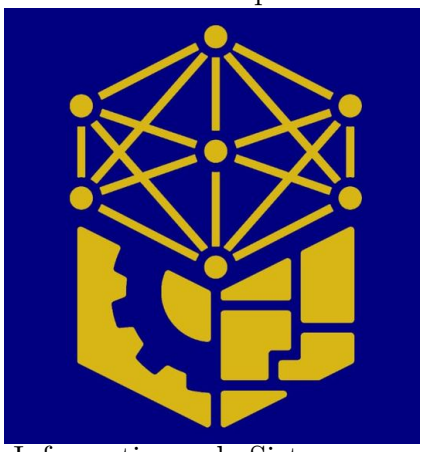
Polo Chura, Marco Rosauro

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

Capítulo 1

Justificación de la propuesta seleccionada

gaga (Theseus OS Project, 2023)

1.1 Nombre del sistema operativo base

1.2 Objetivos del proyecto original

1.3 Arquitectura y enfoque técnico

1.4 Nivel de complejidad y adecuación al contexto educativo

1.5 Comunidad, documentacion y soporte disponible

Capítulo 2

Argumentos técnicos

2.1 Compatibilidad con herramientas de desarrollo

2.2 Modularidad y posibilidad de adaptación

2.3 Lenguaje de programación utilizado

2.4 Facilidad de compilación, prueba y depuración

2.5 Escalabilidad para futuras extensiones

Capítulo 3

Argumentos pedagógicos

3.1 Claridad conceptual y didáctica

3.2 Potencial para fomentar el aprendizaje activo

3.3 Relación con los contenidos del curso

3.4 Posibilidad de trabajo colaborativo y evaluación progresiva

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, 2025). 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.

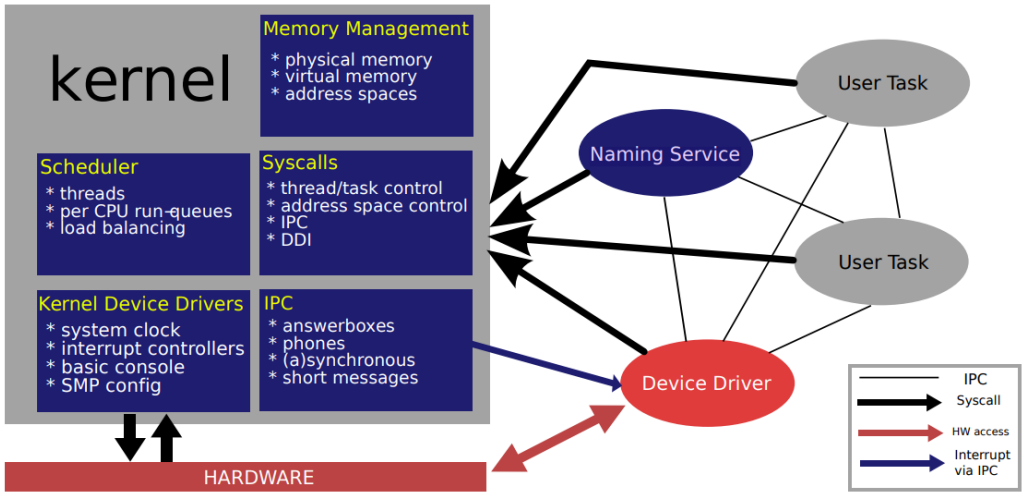


Figura 4.1: Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2006)).

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, respon-

sables 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, diversas primitivas de sincronización interna y técnicas de bloqueo de grano fino, con el objetivo de mejorar el rendimiento y la escalabilidad del sistema.

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).

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)

4.2.5 Sistema de archivos

El sistema operativo HelenOS, tiene el soporte de sistemas de archivos se basa 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).

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).

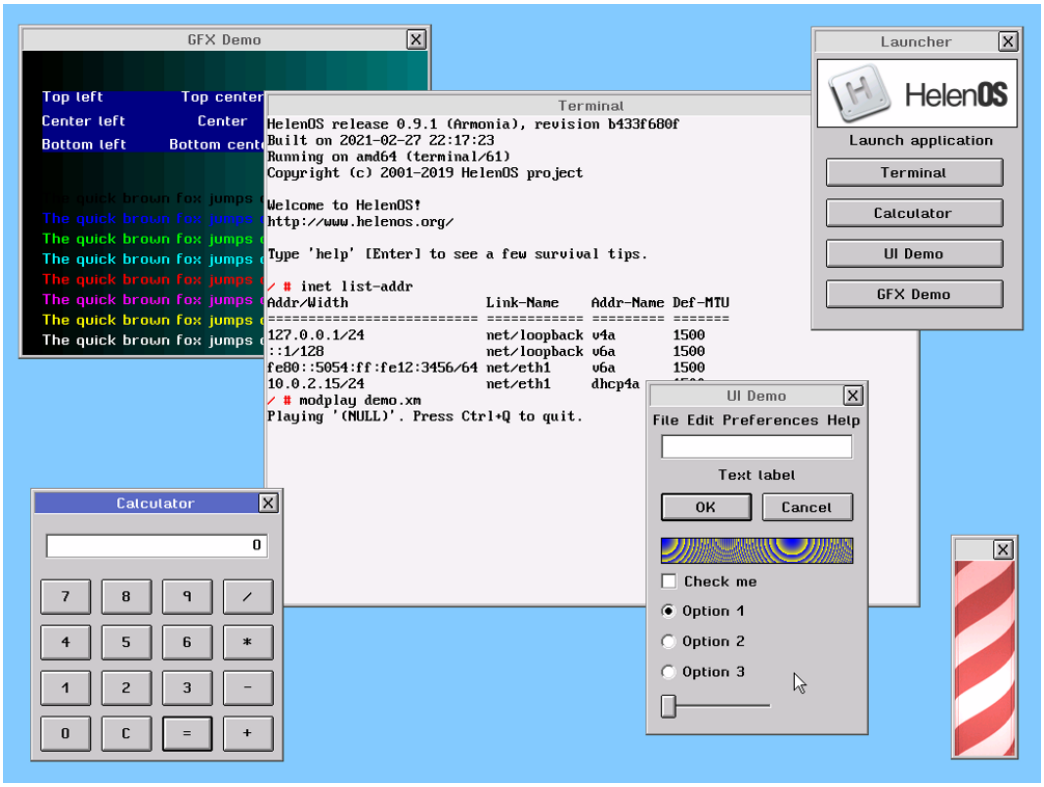


Figura 4.2: Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2026)).

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).

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.

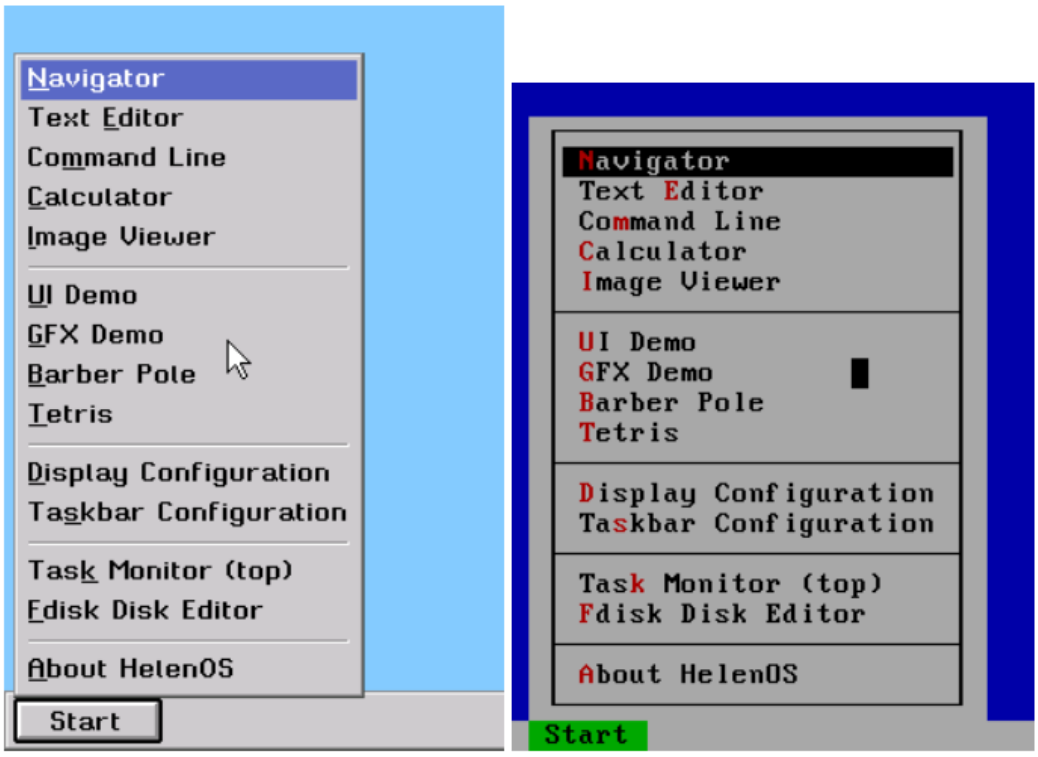


Figura 5.1: Interfaz de usuario. (fuente: (HelenOS Project, 2026)).

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).

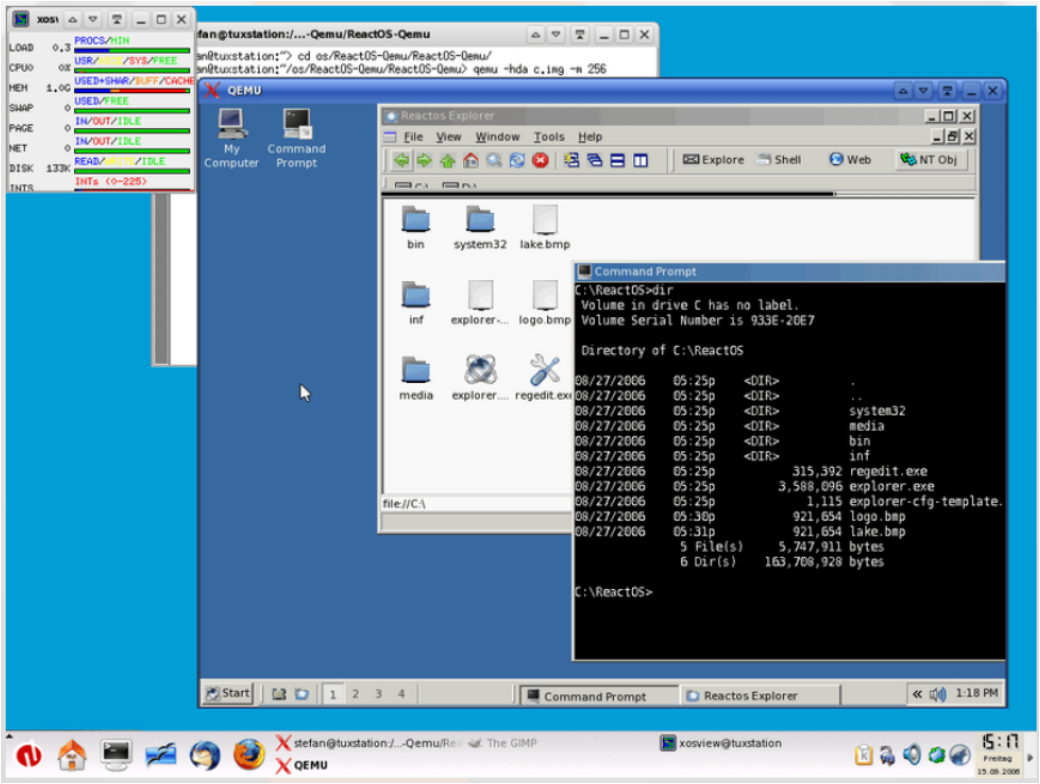


Figura 5.2: QEMU es un emulador y virtualizador de código abierto capaz de realizar emulación de sistema completo en múltiples arquitecturas (fuente: (HelenOS Project, 2026)).

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 Tentativo de Implementación (5 Semanas)

Semana	**Actividades Principales**	**Descripción Ampliada**	**Componentes Involucrados**
1	Instalación y configuración del entorno; Compilación inicial; Validación en QEMU	Se prepara el entorno completo de desarrollo de HelenOS. Esto incluye la instalación del compilador cruzado, la descarga del código fuente oficial, la verificación del sistema de construcción y la ejecución de la primera compilación completa. Finalmente, se valida la ejecución del sistema en QEMU, lo cual asegura que el pipeline de desarrollo esté correctamente configurado desde el inicio.	Toolchain; Build System; QEMU

Continúa en la siguiente página

Semana	Actividades Principales	Descripción Ampliada	Componentes Involucrados
2	Revisión de arquitectura; Selección de módulos; Diseño técnico preliminar	Se analiza la arquitectura de HelenOS, centrada en su microkernel, el mecanismo IPC, la administración de tareas y el VFS modular. Se seleccionan los módulos a implementar o adaptar, y se elabora el diseño técnico inicial, incluyendo diagramas de interacción, estructuras de datos, interfaces y puntos de extensión que permitirán una integración progresiva durante las siguientes semanas.	Kernel; IPC; VFS; Diseño de Software
3	Implementación del Kernel; Scheduler básico; Pruebas unitarias	Se desarrollan las funciones internas del kernel relacionadas con la administración de procesos y se implementa un scheduler básico (por ejemplo, Round Robin). En paralelo, se construyen pruebas unitarias para validar módulos críticos, con el fin de garantizar estabilidad previa a la integración con otros subsistemas.	Kernel; Gestión de Procesos; Pruebas Unitarias

Continúa en la siguiente página

Semana	Actividades Principales	Descripción Ampliada	Componentes Involucrados
4	Implementación del VFS mínimo; Integración Kernel–VFS; Comandos en Shell	Se implementa un sistema de archivos virtual mínimo para soportar operaciones esenciales como lectura, escritura y montaje. Luego, se integran las llamadas del kernel hacia el VFS para permitir interacción estructurada con el almacenamiento. Finalmente, se incorporan comandos básicos en la shell del sistema para validar directamente el funcionamiento del VFS dentro de HelenOS.	VFS; Kernel; Shell

Continúa en la siguiente página

Semana	Actividades Principales	Descripción Ampliada	Componentes Involucrados
5	Pruebas de integración; Pruebas de regresión; Documentación final	Se ejecutan pruebas integrales del sistema operativo, evaluando la interoperabilidad entre componentes, el manejo de recursos, la estabilidad del scheduler y la consistencia del sistema de archivos. Además, se realizan pruebas de regresión contra versiones previas para asegurar que no existan degradaciones. La semana culmina con la elaboración completa de la documentación técnica final, incluyendo diagramas, decisiones arquitectónicas, resultados de pruebas y conclusiones.	Todos los componentes

6.2 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.

6.3 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
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

Continúa en la siguiente página

Riesgo	Descripción técnica	Impacto	Probabilidad	Mitigación	Indicadores tempranos
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
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

Continúa en la siguiente página

Riesgo	**Descripción técnica**	**Impacto**	**Probabilidad**	**Mitigación**	**Indicadores tempranos**
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. (2025). 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.

Theseus OS Project (2023). The theseus os book. Recuperado de https://www.theseus-os.com/Theseus/book/index.html.

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