

# **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 Informática y de Sistemas

Asesor: Ugarte Rojas, Héctor Eduardo

Memorial Universidad Nacional San Antonio Abad del Cusco

Semestre 2025-II

Cusco

Perú

# Índice general

|   |           |
|---|-----------|
| <b>Índice de tablas</b>   | <b>v</b>  |
| <b>Índice de figuras</b>  | <b>vi</b> |
| <b>1 Justificación de la propuesta seleccionada</b>                   | <b>2</b>  |
| 1.1 Nombre del sistema operativo base . . . . .                       | 3         |
| 1.2 Objetivos del proyecto original . . . . .                         | 3         |
| 1.3 Arquitectura y enfoque técnico . . . . .                          | 3         |
| 1.4 Nivel de complejidad y adecuación al contexto educativo . . . . . | 3         |
| 1.5 Comunidad, documentación y soporte disponible . . . . .           | 3         |
| <b>2 Argumentos técnicos</b>  | <b>4</b>  |
| 2.1 Compatibilidad con herramientas de desarrollo . . . . .           | 4         |
| 2.2 Modularidad y posibilidad de adaptación . . . . .                 | 4         |
| 2.3 Lenguaje de programación utilizado . . . . .                      | 4         |
| 2.4 Facilidad de compilación, prueba y depuración . . . . .           | 4         |
| 2.5 Escalabilidad para futuras extensiones . . . . .                  | 4         |
| <b>3 Argumentos pedagógicos</b>                                       | <b>5</b>  |

|          |   |           |
|----------|---|-----------|
| 3.1      | Claridad conceptual y didáctica . . . . .                             | 5         |
| 3.2      | Potencial para fomentar el aprendizaje activo . . . . .               | 5         |
| 3.3      | Relación con los contenidos del curso . . . . .                       | 5         |
| 3.4      | Posibilidad de trabajo colaborativo y evaluación progresiva . . . . . | 5         |
| <b>4</b> | <b>Diseño del sistema operativo propuesto</b>                         | <b>6</b>  |
| 4.1      | Diagrama de arquitectura general . . . . .                            | 7         |
| 4.2      | Componentes a implementar . . . . .                                   | 8         |
| 4.2.1    | Bootloader . . . . .  | 8         |
| 4.2.2    | Kernel básico . . . . .   | 9         |
| 4.2.3    | Gestión de procesos . . . . .   | 11        |
| 4.2.4    | Gestión de memoria . . . . .  | 12        |
| 4.2.5    | Sistema de archivos . . . . .   | 15        |
| 4.2.6    | Interfaz de usuario . . . . .   | 18        |
| 4.2.7    | Políticas de planificación y manejo de recursos . . . . .             | 19        |
| 4.2.8    | Flujo de ejecución básico . . . . .                                   | 22        |
| <b>5</b> | <b>Herramientas y entorno de desarrollo</b>                           | <b>23</b> |
| 5.1      | Lenguaje(s) de programación alto y bajo nivel . . . . .               | 23        |
| 5.2      | Compilador cruzado . . . . .  | 23        |
| 5.3      | Emulador . . . . .  | 23        |
| 5.4      | Control de versiones (Git) . . . . .                                  | 23        |
| 5.5      | Editor o entorno de desarrollo . . . . .                              | 23        |
| <b>6</b> | <b>Planificación de implementación</b>                                | <b>24</b> |

|                    |   |           |
|--------------------|---|-----------|
| 6.1                | Cronograma tentativo por componentes. . . . . | 24        |
| 6.2                | Estrategia de pruebas y validación. . . . .   | 24        |
| 6.3                | Posibles riesgos y cómo mitigarlos. . . . .   | 24        |
| <b>Referencias</b> |   | <b>25</b> |

# Índice de tablas

|     |  |    |
|-----|--|----|
| 4.1 | Subsistemas fundamentales del kernel de HelenOS . . . . .            | 10 |
| 4.2 | Aspectos esenciales de la gestión de memoria en HelenOS . . . . .    | 13 |
| 4.3 | Componentes principales del subsistema de archivos en HelenOS . . .  | 16 |
| 4.4 | Políticas de planificación y manejo de recursos en HelenOS . . . . . | 20 |

# Índice de figuras

|  |    |
|--|----|
| 4.1 Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2006)) . . . . . | 8  |
| 4.2 Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2026)) . . . . . | 19 |

# 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 → Ethernet → IP → 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.

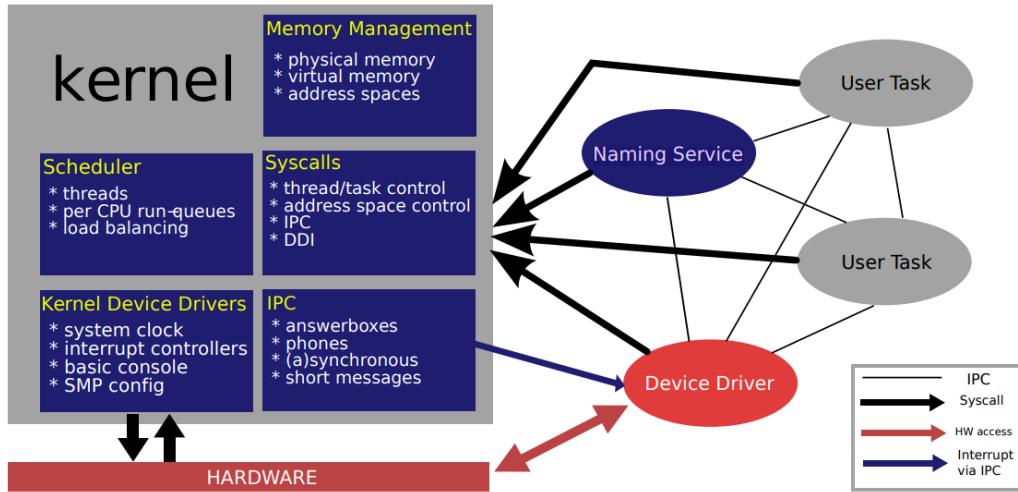


Figura 4.1: Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2006)).

## 4.2 Componentes a implementar

### 4.2.1 Bootloader

En la arquitectura x86-64, HelenOS emplea el gestor de arranque GRUB (Grub bootloader). 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 Mejdréch, Lukáš and others, 2006).

#### 4.2.2 Kernel básico

Según (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 He-  
lenOS

| <b>Subsistema del Kernel</b> | <b>Entidades / Mecanismos</b>  | <b>Función Principal</b>  |
|------------------------------|--|---|
| Programación (Scheduling)    | Hilos del kernel, hilos de usuario, pseudohilos, tareas, colas de ejecución multivel, 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)   |
| 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 |

Tabla 4.1 – continuación de la página anterior

| Subsistema del Kernel             | Entidades / Mecanismos  | Función Principal   |
|-----------------------------------|---|---|
| Comunicación entre Procesos (IPC) | Mensajes cortos síncronos/asíncronos, <i>teléfonos</i> (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.

**Commutación de contexto.** La commutación 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 | <p>La memoria física se organiza en zonas contiguas que contienen marcos (<i>frames</i>) administrados mediante un <i>buddy system</i>. Cada zona mantiene contadores de marcos libres y ocupados.</p> | <p>Permitir asignación eficiente de bloques contiguos y escalabilidad del sistema de memoria física.</p> |
| Asignador de marcos    | <p>El <i>frame allocator</i> gestiona solicitudes de memoria en potencias de dos dentro de una zona, utilizando el <i>buddy allocator</i>.</p>   | <p>Asignación rápida y consistente de memoria física alineada.</p>                                       |
| Slab allocator         | <p>Asignador de objetos pequeños y frecuentes basado en <i>slabs</i> y <i>magazines</i> por CPU, evitando bloqueos globales.</p>   | <p>Reducir fragmentación interna y mejorar concurrencia y rendimiento del kernel.</p>                    |

Tabla 4.2 – continuación

| Componente              | Descripción técnica   | Objetivo / Beneficio   |
|-------------------------|---|--|
| 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.                         |
| Compartición de memoria | Las áreas pueden compartirse entre tareas mediante estructuras de referencia y un <i>pagemap</i> 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.                     |

Tabla 4.2 – continuación

| Componente       | Descripción técnica   | Objetivo / Beneficio   |
|------------------|---|--|
| TLB y coherencia | El sistema mantiene coherencia entre TLB y tablas de páginas mediante invali-daciones y <i>TLB shootdown</i> en sistemas SMP. | Garantizar consisten-cia y corrección en ejecución multiproce-sador. |

#### 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 al-macenamiento en un entorno multiservidor (Zárevúcky, Jiří, 2012; HelenOS Project, 2006).

Tabla 4.3: Componentes principales del subsistema de archivos en HelenOS

| <b>Componente</b>                  | <b>Ubicación / Tipo</b>                         | <b>Función principal</b>  |
|------------------------------------|---|---|
| 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  |
| Frontend VFS                       | Módulo interno del servidor VFS                 | Atiende llamadas básicas de aplicaciones ( <code>open</code> , <code>read</code> , <code>write</code> , <code>close</code> ), 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   |

Tabla 4.3 – continuación

| <b>Componente</b>                       | <b>Ubicación / Tipo</b>                            | <b>Función principal</b>  |
|---|--|---|
| Capa de compatibilidad POSIX            | Biblioteca <code>libc</code> 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        |
| 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).

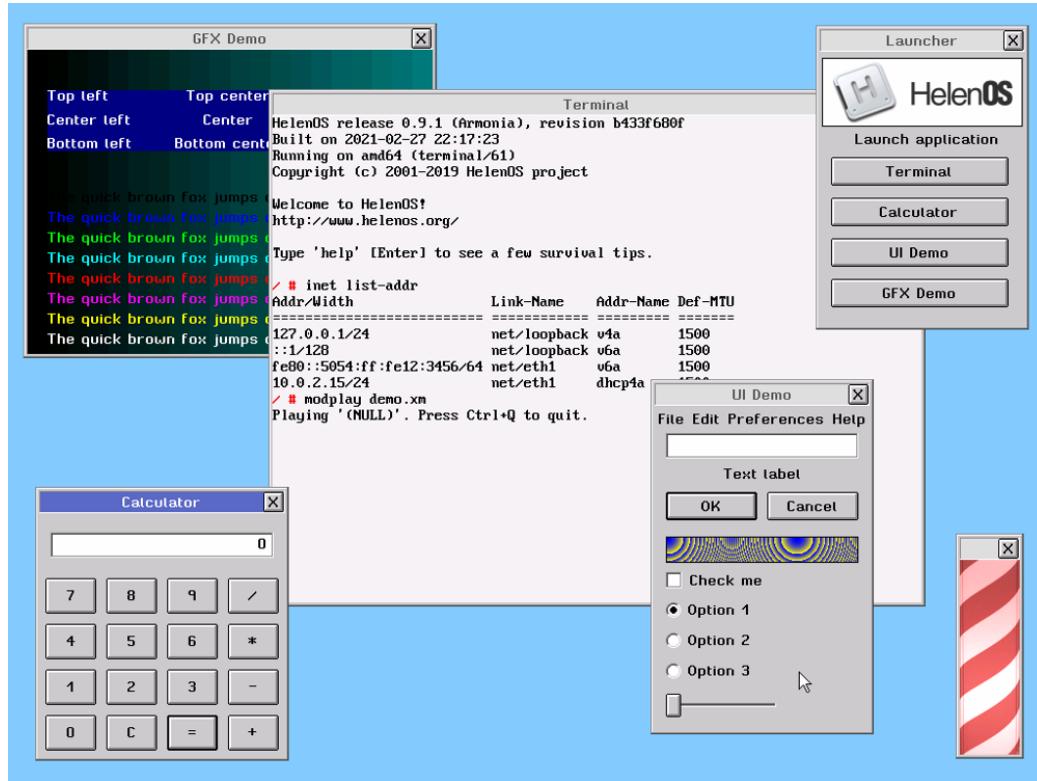


Figura 4.2: Diseño general de HelenOS y organización multiservidor basada en microkernel (fuente: (HelenOS Project, 2026)).

#### 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).

### 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. |

Tabla 4.4 – continuación

| Política / Mecanismo                 | Descripción técnica (resumen)  | Objetivo / Beneficios  |
|--------------------------------------|--|--|
| 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. |
| 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.               |

Tabla 4.4 – continuación

| Política / Mecanismo             | Descripción técnica (resumen)  | Objetivo / Beneficios                                     |
|----------------------------------|--|---|
| 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.

#### 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**

**5.1 Lenguaje(s) de programación alto y bajo nivel**

**5.2 Compilador cruzado**

**5.3 Emulador**

**5.4 Control de versiones (Git)**

**5.5 Editor o entorno de desarrollo**

# **Capítulo 6**

## **Planificación de implementación**

**6.1 Cronograma tentativo por componentes.**

**6.2 Estrategia de pruebas y validación.**

**6.3 Posibles riesgos y cómo mitigarlos.**

# 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 Mejdréch, 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**