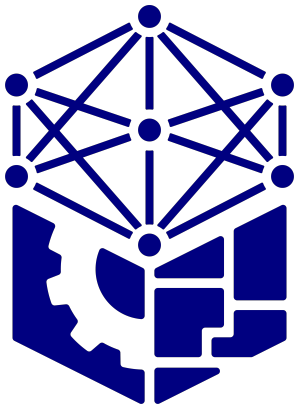

Proyecto de Investigación: Análisis, Diseño y Construcción de un Sistema Operativo desde Cero

by

© Magaña Osorio Jhoel Fabrizzio

© Colque Quispe Fidel Enrique

© Montalvo Solórzano Rosy Aurely

© Quispe Llavilla Jhon Andherson

Trabajo de investigación de la asignatura de Sistemas Operativos

Docente: *Ugarte Rojas Hector Ugarte*

Universidad Nacional de San Antonio Abad del Cusco

Facultad de Ingeniería Eléctrica, Electrónica, Informática y Mecánica

Escuela Profesional de Ingeniería Informática y de Sistemas

Cusco - Perú

2025

Contents

List of Tables

List of Figures

Chapter 1

Introducción

1.1 Proposito de la Investigación

El presente documento forma parte del segundo entregable del proyecto semestral correspondiente a la asignatura de Sistemas Operativos. El propósito de este proyecto es investigar y analizar diferentes propuestas y proyectos reales de creación de sistemas operativos desde cero, identificando sus enfoques, herramientas, arquitecturas y objetos pedagógicos o técnicos.

El presente documento busca ofrecer una visión comparativa y reflexiva sobre las distintas aproximaciones existentes, con el fin de seleccionar posteriormente una base a decuada para la implementación de un sistema operativo.

1.2 Criterios de selección

La selección de las propuestas analizadas se realizó bajo criterios técnicos, pedagógicos y de accesibilidad, con el propósito de abarcar un conjunto amplio y diverso de sistemas

operativos desarrollados desde cero. Los principales criterios fueron los siguientes:

- **Disponibilidad del código fuente y documentación:** se priorizaron proyectos de código abierto con repositorios activos y guías técnicas verificables.
- **Variedad de arquitecturas y lenguajes:** se buscó incluir proyectos basados en lenguajes clásicos y otros lenguajes modernos, representando distintos paradigmas de diseño.
- **Arquitectura** se contemplaron modelos monolíticos, microkernel e híbridos, a fin de comparar enfoques estructurales.
- **Nivel de complejidad y accesibilidad:** los sistemas seleccionados presentan distintos grados de dificultad, desde proyectos introductorios hasta desarrollos avanzados.

1.3 Metodología

La metodología adoptada en la presente investigación es de carácter documental, comparativa y aplicada, orientada no solo a la recopilación y análisis de información técnica, sino también a la verificación práctica del funcionamiento de los sistemas operativos seleccionados. Este enfoque busca integrar el estudio teórico con la experimentación directa.

El desarrollo metodológico se estructuró en las siguientes fases:

1. **Investigación bibliográfica y exploratoria:** Se llevó a cabo una búsqueda exhaustiva de fuentes especializadas, tales como **repositorios oficiales, artículos**

científicos, blogs técnicos, manuales de desarrollo y documentación de proyectos open source. Esta revisión permitió identificar iniciativas relevantes de creación de sistemas operativos desde cero.

2. **Análisis técnico y estructural de los proyectos:** En esta fase se estudió la **arquitectura interna, los componentes principales (kernel, gestor de memoria, sistema de archivos, interfaz, etc.) y los lenguajes de implementación** de cada propuesta.

3. **Implementación y experimentación:**

   Los proyectos seleccionados fueron **descargados, compilados y ejecutados en entornos controlados** utilizando herramientas como **QEMU, VirtualBox o VMware**, con el objetivo de observar su comportamiento real. Esta etapa permitió comprobar la **viabilidad, estabilidad y compatibilidad** de cada sistema operativo, así como analizar sus requerimientos de hardware y dependencias de compilación.

4. **Sistematización y comparación de resultados:** La información teórica y los resultados experimentales se organizaron en **tablas comparativas, gráficos y esquemas** que permiten una evaluación integral. Este proceso facilitó la identificación de **patrones comunes**, diferencias arquitectónicas y niveles de accesibilidad entre los distintos sistemas.

1.4 Propuestas Analizadas

1.4.1 HelenOS

1.4.1.1 Nombre del proyecto o sistema operativo

HelenOS es un sistema operativo multiserver basado en un microkernel, desarrollado completamente desde cero y sin dependencia de Unix. Su diseño busca la modularidad, la portabilidad y la tolerancia a fallos, orientado principalmente a la investigación académica y la enseñanza de arquitectura de sistemas operativos (?, ?).

1.4.1.2 Enlace al repositorio y/o documentación oficial

Repositorio oficial: `https://github.com/HelenOS/helenos`.

Sitio web del proyecto: `https://www.helenos.org`.

Documentación técnica (PDF): `https://www.helenos.org/doc/prjdoc.pdf`

1.4.1.3 Objetivo del proyecto

El objetivo de HelenOS es principalmente **educativo y experimental**. Su propósito es servir como plataforma de estudio de sistemas operativos modernos, promoviendo un enfoque modular donde cada servicio (sistema de archivos, red, controladores o interfaz gráfica) se implementa como un servidor independiente en espacio de usuario. Este diseño fomenta el análisis de conceptos avanzados como la comunicación entre procesos (IPC) y la separación entre el núcleo y los servicios de usuario (?, ?).

1.4.1.4 Lenguaje de implementación

El proyecto está desarrollado principalmente en el lenguaje **C**, con partes críticas escritas en **ensamblador** (para el arranque del sistema y la gestión de interrupciones). Se mantiene compatibilidad con los estándares **C11** y **C++14**, facilitando la portabilidad y la integración de nuevos componentes (?, ?).

1.4.1.5 Arquitectura del sistema

HelenOS adopta una arquitectura **microkernel multiserver**. El microkernel proporciona únicamente los servicios esenciales —planificación, gestión de memoria, interrupciones e IPC— mientras que todos los demás servicios (como el sistema de archivos, la red o la interfaz gráfica) se ejecutan como procesos de usuario independientes que se comunican mediante paso de mensajes (*message passing*) (?, ?).

1.4.1.6 Componentes implementados

- **Gestión de procesos e hilos:** Soporta multitarea con planificación por prioridades y aislamiento entre procesos. El modelo de ejecución permite múltiples hilos por proceso y comunicación sincronizada entre ellos mediante IPC.
- **Gestión de memoria:** Implementa un sistema de memoria virtual con paginación y protección por espacio de direcciones. El kernel se encarga de la asignación básica y los servidores de usuario gestionan asignadores dinámicos (*slab allocators*) y regiones compartidas.
- **Sistema de archivos (VFS):** Se basa en una arquitectura de VFS (Virtual

File System) donde cada tipo de sistema de archivos (por ejemplo, FAT, EXT2) se implementa como un servidor separado en espacio de usuario. Esto permite montar y desmontar sistemas de archivos sin reiniciar el núcleo.

- **Dispositivos y controladores:** Los *drivers* se ejecutan fuera del núcleo (user-space drivers), lo que evita que los errores en controladores afecten la estabilidad general del sistema. Hay soporte para dispositivos de almacenamiento, red, USB, video, teclado y mouse.

- **Sistema de red:** Incluye una pila TCP/IP modular con soporte para IPv4, interfaces Ethernet virtuales, y utilidades de diagnóstico como `ping`, `netstat`, `inet list-addr` y `arp`.

- **Sistema gráfico:** Dispone de un entorno gráfico de ventanas (GUI) que corre completamente en espacio de usuario. Proporciona aplicaciones nativas como *Terminal*, *Text Editor*, *Navigator*, *Calculator*, *GFX Demo* y *Tetris*. El gestor de ventanas se comunica con el servidor gráfico a través del protocolo `display-server`.

- **Comunicación e IPC:** El núcleo ofrece un sistema de comunicación entre procesos mediante intercambio de mensajes (*message passing*) orientado a puertos. Este mecanismo es la base de toda interacción entre servidores y aplicaciones.

- **Seguridad y aislamiento:** Cada componente ejecuta con privilegios mínimos. Si un servidor falla, el kernel lo aísla y otros componentes pueden seguir ejecutándose, lo que mejora la tolerancia a fallos.

- **Portabilidad:** El código fuente está diseñado para ser independiente de la arquitectura. Actualmente soporta múltiples plataformas: `ia32`, `amd64`, `arm32`, `mips32`, `ppc32`, `sparc64` e incluso `ia64` (Itanium).

1.4.1.7 Herramientas utilizadas

El proceso de compilación y prueba de HelenOS requiere la preparación de un entorno de desarrollo cruzado (*cross-compiling*) que permite generar imágenes del sistema para múltiples arquitecturas. A continuación se detallan las principales herramientas utilizadas:

- **Toolchain cruzado:** Se genera mediante el script `tools/toolchain.sh`, que compila una cadena completa de herramientas `gcc`, `binutils` y `gdb` adaptadas para la arquitectura destino (por ejemplo, `ia64-helenos-gcc`). El entorno de compilación se instala por defecto en el directorio `/usr/local/cross` y puede personalizarse mediante la variable `CROSS_PREFIX`.

- **Meson y Ninja:** HelenOS utiliza el sistema de construcción **Meson** junto al generador **Ninja**, lo que permite una configuración modular y compilaciones rápidas e incrementales.

- **Dependencias del entorno:** Para distribuciones basadas en Debian/Ubuntu, se necesitan los paquetes `build-essential`, `wget`, `texinfo`, `flex`, `bison`, `dialog`, `python3-yaml` y `genisoimage`.

- **Configuración del proyecto:** El script `configure.sh` permite definir perfiles preestablecidos de compilación. Por ejemplo, para la arquitectura `amd64`:

```
git clone https://github.com/HelenOS/helenos.git
mkdir -p build/amd64
cd build/amd64
../../helenos/configure.sh amd64
ninja
ninja image_path
```

- **Emulación y prueba:** HelenOS puede ejecutarse en **QEMU** o **VirtualBox**. Para iniciar el sistema en QEMU se recomienda:

```
qemu-system-x86_64 -m 512 -cdrom image.iso \
-usb -device usb-tablet -serial mon:stdio -display sdl
```

Además, el repositorio incluye scripts auxiliares en `tools/ew.py` que automatizan el proceso de arranque en QEMU, y configuraciones predefinidas en `tools/conf` para ajustar parámetros de memoria o dispositivos virtuales.

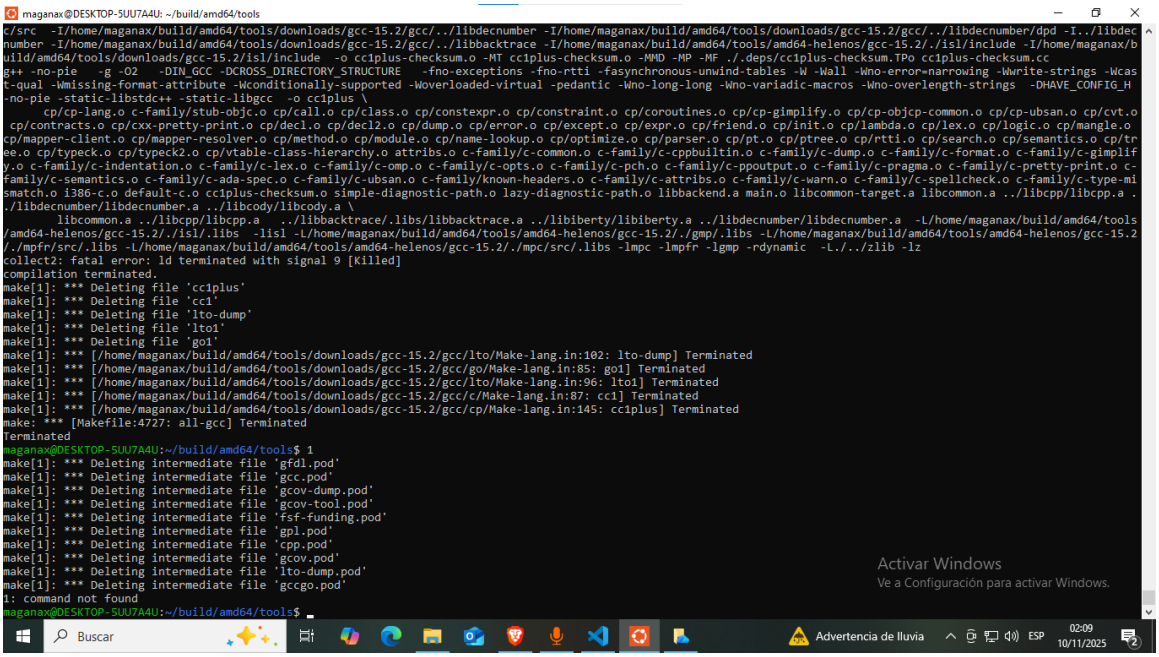

Figure 1.1: Elaboración Propia. Compilación cruzada del toolchain.sh

1.4.1.8 Nivel de complejidad y accesibilidad

El nivel de complejidad de HelenOS es alto, ya que su comprensión requiere conocimientos de compilación cruzada, IPC y gestión de memoria. Sin embargo, su estructura modular y bien documentada lo convierte en una excelente herramienta didáctica donde se pueden analizar de manera aislada servicios específicos como el sistema de archivos o el entorno gráfico (?, ?).

Table 1.1: Ficha técnica resumida de HelenOS.

Diseño	Microkernel multiserver modular
Lenguajes	C (principal) y ensamblador (arranque y bajo nivel)
Arquitecturas	ia32, amd64, arm32, mips32, ppc32, sparc64
Componentes clave	Kernel mínimo, VFS, red, GUI, drivers en espacio de usuario
Herramientas	GCC (toolchain cruzado), Meson, Ninja, QEMU/VirtualBox
Uso educativo	Alto: análisis modular de componentes y comunicación entre procesos

Nota. Adaptado de "About HelenOS" (HelenOS Project, 2024)

1.4.2 Visopsys

1.4.2.1 Nombre del proyecto o sistema operativo

Visopsys (*Visual Operating System*) es un sistema operativo alternativo, gratuito y desarrollado desde cero por Andy McLaughlin desde 1997. A diferencia de otros sistemas, no está basado en Unix ni Linux, sino que implementa su propio kernel, gestor de archivos y entorno gráfico. Está diseñado para ser compacto, rápido y con una interfaz visual moderna, integrando tanto funcionalidades de línea de comandos como de entorno gráfico (?, ?, ?).

1.4.2.2 Enlace al repositorio y/o documentación oficial

Repositorio oficial y descargas: `https://sourceforge.net/projects/visopsys/files/`

Sitio web del proyecto: `https://visopsys.org`

Presentación técnica (Scribd): `https://es.scribd.com/presentation/619552414/SO-visopsys`

1.4.2.3 Objetivo del proyecto

El objetivo de Visopsys es **educativo y experimental**. Busca proporcionar una plataforma funcional que demuestre los principios fundamentales del diseño de sistemas operativos, la gestión de procesos, la memoria y los sistemas de archivos. Está enfocado en ser simple, visual y completamente entendible, lo que lo hace ideal para el estudio de arquitecturas operativas en cursos de ingeniería de sistemas o informática (?, ?).

1.4.2.4 Lenguajes de implementación

El sistema está implementado principalmente en el lenguaje **C**, con secciones críticas escritas en **ensamblador x86**. Esta combinación le permite optimizar la comunicación con el hardware y mantener un tamaño reducido del núcleo y del sistema base (?, ?).

1.4.2.5 Arquitectura del sistema

Visopsys implementa una arquitectura **monolítica modular**. Su núcleo controla directamente los procesos, la gestión de memoria, el sistema de archivos y los controladores de dispositivos, aunque permite la carga dinámica de módulos y

extensiones en tiempo de ejecución. Esto le otorga un balance entre rendimiento, simplicidad y capacidad de expansión (?, ?).

1.4.2.6 Componentes implementados

- **Gestión de procesos e hilos:** Visopsys soporta multitarea cooperativa y por prioridades, con un planificador de procesos básico implementado en su núcleo. Cada tarea dispone de su propio contexto de ejecución y control de interrupciones.

- **Gestión de memoria:** Implementa un sistema de memoria virtual simple con asignación dinámica y control directo del espacio de direcciones. El sistema garantiza aislamiento básico entre tareas activas.

- **Sistema de archivos:** Integra su propio sistema VFS con soporte para FAT12, FAT16 y FAT32, además de detección de tablas de particiones MBR y GPT. Incluye herramientas gráficas para formateo, copia, borrado y verificación de discos.

- **Dispositivos y controladores:** Dispone de controladores nativos para discos IDE y SATA, tarjetas gráficas básicas (modo VESA), teclado, mouse y almacenamiento USB. Los drivers se integran directamente en el kernel.

- **Interfaz gráfica (GUI):** Ofrece un entorno gráfico funcional con ventanas, menús y una barra de herramientas principal. Incluye aplicaciones integradas como un administrador de archivos, visor hexadecimal, editor de texto y panel de control del sistema.

- **Herramientas del sistema:** Contiene utilidades para crear, copiar o gestionar particiones, visualizar la memoria y monitorear el rendimiento del CPU en tiempo real.
- **Soporte de hardware:** El sistema está diseñado para funcionar en procesadores x86 y x86-64, aunque se ejecuta principalmente en entornos de 32 bits para mantener compatibilidad amplia.

1.4.2.7 Herramientas utilizadas

Visopsys puede compilarse con **GCC** en sistemas Linux, BSD o Windows mediante entornos cruzados. También se distribuye como una **ISO ejecutable** que puede probarse directamente sin instalación.

- **Compilador:** GCC y utilidades estándar de GNU (`make`, `binutils`).
- **Entorno de prueba:** Compatible con **VirtualBox** y **QEMU**.
- **Configuración sugerida:**

```
Sistema: Other/Unknown (32-bit)
Memoria: 256 MB
Arranque: visopsys-0.92.iso
Desactivar EFI y habilitar CD-ROM
```

- **Ejecución:** Al iniciar desde la ISO, carga automáticamente el entorno gráfico sin requerir instalación.

Figure 1.2: Implementación de Visopsys en VirtualBox.

1.4.2.8 Nivel de complejidad y accesibilidad

El nivel de complejidad de Visopsys es **medio**, ya que su código fuente es legible y su estructura modular facilita el aprendizaje. Permite explorar los conceptos fundamentales de sistemas operativos, como la planificación, memoria, drivers y GUI, dentro de un único entorno funcional y visual (?, ?).

Table 1.2: Ficha técnica resumida de Visopsys.

Diseño	Monolítico modular con GUI integrada
Lenguajes	C y ensamblador (x86)
Arquitecturas	x86 (32 bits)
Componentes clave	Kernel, VFS, GUI, gestor de procesos, controladores IDE/SATA, herramientas gráficas
Herramientas	GCC, Make, VirtualBox/QEMU
Uso educativo	Medio: análisis de kernel, GUI y gestión de memoria

Nota. Adaptado de "SO Visopsys" (McLaughlin, 2023).

1.4.3 ToarusOS

1.4.3.1 Nombre del proyecto o sistema operativo

ToaruOS es un sistema operativo independiente, escrito casi completamente desde cero, diseñado como un recurso educativo que ofrece una implementación completa de un SO moderno.

1.4.3.2 Enlace al repositorio y/o documentación oficial

Repositorio oficial: `https://github.com/klange/toaruos`

Sitio web del proyecto: `http://toaruos.org/`

1.4.4 Objetivo del proyecto

ToarusOS esta diseñado para servir como recurso educativo que permita estudiar y comprender el funcionamiento de un sistema operativo completo. Pretendiendo ser una representación compacta y funcional de los componentes que forman parte de un sistema operativo de escritorio moderno.

1.4.4.1 Lenguajes de implementación

ToaruOS está implementado principalmente en el lenguaje de programación **C**, además de ello, se encuentra implementado con el lenguaje de programación de Kuroko, un lenguaje propio, un lenguaje de programación dinámico, compilado a bytecode y un dialecto de Python.

1.4.4.2 Arquitectura del sistema

ToaruOS adopta una arquitectura que posee un kernel híbrido y modular, denominada Misaka, es una arquitectura híbrida con módulos cargables en tiempo de ejecución. El kernel proporciona servicios esenciales como la gestión de procesos, memoria y sistemas de archivos, mientras que otros componentes del sistema operativo, como el entorno gráfico y las aplicaciones, se ejecutan en espacio de usuario.

1.4.4.3 Componentes implementados

- **Gestión de procesos e hilos:** ToaruOS ToaruOS utiliza el kernel Misaka, un núcleo híbrido con soporte para multiprocesamiento simétrico (SMP), que implementa un planificador de procesos, mecanismos de comunicación entre

procesos mediante el sistema pex y una arquitectura basada en procesos expuestos a través del sistema de archivos virtual `/proc`.

- **Gestión de memoria:** Incorpora un sistema de memoria con paginación y asignación dinámica dentro del kernel, complementado por una implementación propia de la biblioteca estándar de C (libc) que administra la memoria en espacio de usuario, junto con el uso de un ramdisk comprimido como sistema de archivos raíz inicial.

- **Sistema de archivos:** ToaruOS emplea un filesystem raíz montado desde un ramdisk e incluye soporte para ISO9660, además de una estructura tipo Unix que integra directorios virtuales como /dev, /proc y /tmp, permitiendo la interacción con dispositivos, procesos y almacenamiento temporal de forma eficiente.

- **Dispositivos y controladores:** El sistema dispone de un conjunto modular de controladores cargables que gestionan dispositivos gráficos, audio, almacenamiento IDE, red y periféricos de entrada, ofreciendo además integración avanzada con entornos de virtualización como VirtualBox y VMware, mientras continúa el desarrollo de drivers adicionales como AHCI, USB y virtio.

- **Interfaz gráfica (GUI):** ToaruOS cuenta con un entorno gráfico completo basado en Yutani, un servidor de ventanas compositado con aceleración por software y un diseño inspirado en interfaces de finales de los años 2000, acompañado por un conjunto de aplicaciones y herramientas gráficas como la Terminal, el editor Bim y el sistema de widgets TTK.

- **Herramientas de sistema:** Incluye utilidades fundamentales como el shell

Esh con soporte para tuberías y redirecciones, el cargador dinámico ld.so, el editor de texto Bim, el lenguaje de programación Kuroko y un conjunto de comandos Unix-like, complementados por herramientas de desarrollo internas como auto-dep.krk.

- **Soporte de hardware:** ToaruOS funciona principalmente sobre arquitecturas x86-64, con soporte experimental para ARMv8 y compatibilidad comprobada tanto en entornos virtualizados como en hardware real, incorporando funcionalidades específicas para mejorar la experiencia en máquinas virtuales y extendiendo continuamente su compatibilidad mediante el desarrollo de nuevos controladores.

Chapter 2

Dealing with Errors

LaTeX can produce cryptic error messages at times. However, with some experience, it is usually not too difficult to determine what the problem is and how to fix it.

As mentioned earlier, appropriate search terms in Google may help you fix these error messages.