**PROTOTIPO DE SISTEMA OPERATIVO (SO) CON GUI TKINTER**

**YEISSON CAMILO VILLAMIL BLANDÓN**

**MARIO ALEJANDRO PEÑA ARENAS**

**SISTEMAS OPERATIVOS**

**JOSÉ ANTONIO MARTÍNEZ**

**INSTITUCIÓN UNIVERSITARIA DE COLOMBIA**

**BOGOTÁ D.C.**

**SEPTIEMBRE 29 DE 2025**

**INTRODUCCIÓN**

En el presente trabajo se presenta el diseño, implementación y validación de un prototipo educativo de Sistema Operativo (SO) escrito en Python. El prototipo implementa conceptos fundamentales de un SO como gestión de procesos con planificación round-robin (quantum configurable), simulación de memoria con marcos (frames) de tamaño fijo, un sistema de archivos simplificado, mecanismos de sincronización (semáforos) y una interfaz de shell para la ejecución de comandos básicos. Como aporte adicional, se desarrolló una interfaz gráfica de usuario (GUI) con Tkinter que expone el shell, un monitor de procesos en vivo y una visualización del estado de los marcos de memoria, además de accesos rápidos, historial de comandos, tema claro/oscuro y atajos de teclado.

El objetivo principal es consolidar el aprendizaje práctico de los componentes de un SO en un entorno controlado y didáctico, permitiendo la observación directa del comportamiento del planificador y la gestión de memoria. Se describe la arquitectura modular del prototipo, se detallan sus componentes y se proveen instrucciones completas de instalación y ejecución tanto en modo CLI como GUI. Finalmente, se incluyen escenarios de prueba, resultados, consideraciones técnicas, limitaciones y propuestas de trabajo futuro.

**OBJETIVOS**

**OBJETIVO GENERAL**

Diseñar e implementar un prototipo funcional de Sistema Operativo educativo que permita experimentar con gestión de procesos, memoria y entrada/salida, disponible en interfaz de línea de comandos (CLI) y una GUI con Tkinter.

**OBJETIVOS ESPECÍFICOS**

• Implementar un planificador round-robin con quantum configurable y cola de listos.

• Simular memoria principal segmentada en marcos, con asignación/liberación por PID.

• Proveer una shell con comandos básicos (help, ls, cat, write, rm, formatear, run, ps, kill, memstat, exit).

• Desarrollar una GUI con Tkinter que integre el shell, monitor de procesos y visualización de memoria.

• Definir y ejecutar escenarios de prueba para validar el comportamiento del prototipo.

**ALCANCE**

El prototipo cubre conceptos fundamentales y no pretende ser un SO completo. La planificación se realiza con un scheduler simple de round-robin. La memoria se modela con marcos de tamaño fijo y tabla de ocupación por PID. El sistema de archivos es simulado y limitado. La sincronización se exemplifica con semáforos. La GUI se centra en visualización y control didáctico, no en producción.

**REQUERIMIENTOS**

**Requerimientos funcionales**

* El prototipo debe permitir la creación, ejecución y finalización de procesos mediante la CLI y GUI.
* Debe incluir un planificador round-robin con quantum configurable, capaz de alternar procesos en la cola de listos.
* La memoria simulada debe manejar marcos de tamaño fijo, con asignación y liberación según el PID.
* Debe proveer una shell interactiva con comandos básicos, ir a Anexos.
* La GUI debe permitir visualizar:
  + La lista de procesos activos con sus estados.
  + La asignación de marcos de memoria en tiempo real.
  + Historial de comandos ejecutados y accesos rápidos.
* Debe incluir mecanismos de sincronización (semáforos binarios y contables) para coordinar procesos.
* La GUI debe soportar **temas claros y oscuros** y **atajos de teclado** configurables.

**Requerimientos no funcionales**

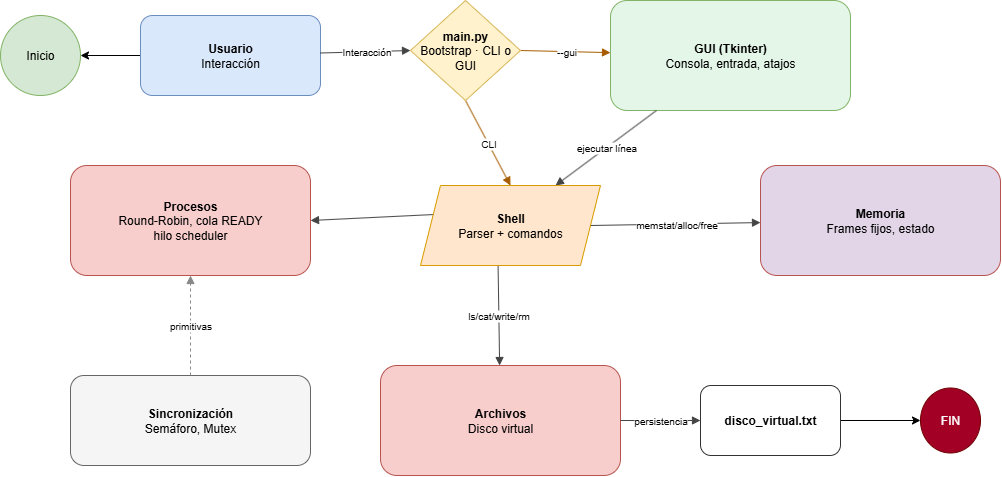
* Debe ejecutarse de forma estable en sistemas Windows, Linux y macOS con Python 3.11 o 3.12.
* La interfaz debe ser intuitiva y responsiva, con actualizaciones periódicas de memoria y procesos (refresco cada 1s) y on-demand.
* Debe ser extensible, permitiendo futuras mejoras como nuevas políticas de planificación, swapping o métricas adicionales.

**Requerimientos técnicos**

* Lenguaje: Python 3.11 o 3.12.
* Interfaz gráfica: Tkinter (incluido en Python estándar; instalar con sudo apt-get install -y python3-tk en Linux si no está disponible).
* Entorno virtual recomendado para instalación de dependencias: venv.
* Versionamiento y colaboración: uso de Git y repositorio remoto en GitHub.
* Dependencias externas: mínimas; solo se permite PyInstaller para generación de ejecutables si se desea.
* Compatibilidad CLI/GUI: el prototipo debe ejecutarse desde línea de comandos (python main.py) o con GUI (python main.py --gui).

**ARQUITECTURA DEL PROTOTIPO**

Diagrama de procesos:



Arquitectura modular:

* Procesos.py: “GestorProcesos” y “Proceso”. Scheduler round-robin, cola de listos, creación y terminación.
* Memoria.py: “Memoria”. Frames fijos, asignación/liberación por PID, lectura/escritura segura, estado.
* Archivos.py: Operaciones de almacenamiento simulado: listar, leer, escribir, eliminar, formatear.
* Sincronizacion.py: Semáforo binario/contable para sincronización básica.
* Shell.py: Interfaz de comandos, mapeo `cmd\_\*`, bucle de lectura y ejecución.
* Gui.py: Interfaz Tkinter (Terminal/Monitor/Memoria), historial, atajos, tema, visualización de frames.
* Main.py: Punto de entrada; `python main.py` (CLI) o `python main.py --gui` (GUI).

**DECISIONES DE DISEÑO**

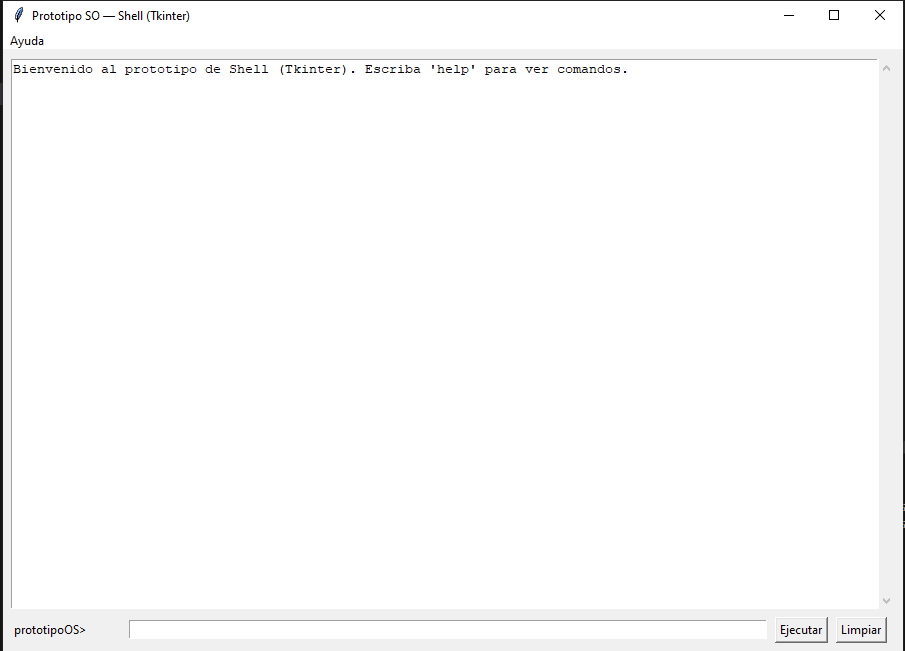
Principales decisiones de diseño:

* Round-robin con quantum pequeño para alternancia perceptible en el monitor.
* Memoria como diccionarios `{frame\_index -> bytearray}` y `{frame\_index -> owner\_pid}` para claridad.
* Shell desacoplada: los comandos llaman a servicios en `procesos`, `memoria` y `archivos`.
* GUI Tkinter que reusa la shell (capturando `stdout/stderr`) y añade vistas de procesos y memoria.
* Refrescado programado de monitor/memoria (cada 1s) + refrescos on-demand tras ejecutar comandos.

**INSTALACIÓN Y EJECUCIÓN**

1. Crear entorno virtual:
   * Windows: `python -m venv .venv && .venv\Scripts\activate`
2. Dependencias
   * Tkinter: en Windows/macOS viene con Python oficial; en Linux: `sudo apt-get install -y python3-tk`  
      - PyInstaller: `pip install pyinstaller`
3. Ejecutar (CLI)
   * `python main.py`
4. Ejecutar (GUI)
   * `python main.py --gui`

**PRUEBAS Y RESULTADOS**

Escenarios de prueba:  
- \*\*Creación y listado de procesos\*\*: `run demo`, `ps` → Verificar aparición en monitor con `PID`, `estado`, `pc`.  
- \*\*Planificación\*\*: Crear múltiples procesos (`run p1`, `run p2`...) y observar round-robin.  
- \*\*Memoria\*\*: Asignar y liberar frames indirectamente a través de procesos; `memstat` y pestaña Memoria.  
- \*\*Archivos\*\*: `write notas.txt "hola"`, `ls`, `cat notas.txt`, `rm notas.txt`, `ls`.  
- \*\*Sincronización\*\*: (si aplica en las rutinas de proceso) probar semáforos con `sincronizacion.py`.  
- \*\*GUI\*\*: Probar atajos (F1, F5, Ctrl+Enter, Ctrl+L), tema oscuro, kill desde Monitor.  
Resultados esperados: la tabla de procesos y el grid de memoria reflejan los cambios en tiempo real o al refrescar.  


**MÉTRICAS Y LIMITACIONES**

Métricas/observables: longitud de cola de listos, quantum, número de frames ocupados/libres, latencia de comandos.  
Limitaciones: sin memoria virtual real, sin I/O real de dispositivos, sistema de archivos simplificado, sin usuarios/ACLs.

**LECCIONES APRENDIDAS**

• Diseñar para visualización acelera el aprendizaje (GUI con monitor y memoria).  
• La separación de responsabilidades (módulos) facilita pruebas y extensibilidad.  
• El round-robin con quantum pequeño produce alternancia clara pero no optimiza rendimiento real.

**CONCLUSIONES**

El prototipo cumple el objetivo educativo al integrar los conceptos principales de un SO en una plataforma ejecutable y observable. La GUI facilita la comprensión de la dinámica del planificador y la gestión de memoria. Si bien no es un SO completo, constituye una base robusta para prácticas, demos y ampliaciones controladas.

**POSIBLES MEJORAS**

• Añadir métricas de rendimiento (tiempos de espera, respuesta, turnaround) por proceso.  
• Implementar prioridades, colas multinivel o SJF para comparar políticas de planificación.  
• Agregar swapping/paginación simulada con fallos de página controlados.  
• Integrar perfiles de proceso parametrizables (CPU-bound, I/O-bound).  
• Persistir métricas y trazas para análisis posterior.

**ANEXOS**

Comandos disponibles:

* ayuda Muestra la lista de comandos disponibles y su descripción.
* borrar Elimina un archivo del disco virtual. Uso: borrar <archivo>
* cat Muestra el contenido de un archivo. Uso: cat <archivo>
* crearproceso Crea y ejecuta un proceso de ejemplo. Uso: crearproceso <nombre\_proceso>
* ejecutar Crea y ejecuta un proceso de ejemplo. Uso: ejecutar <nombre\_proceso>
* eliminar Elimina un archivo del disco virtual. Uso: eliminar <archivo>
* escribir Crea o sobrescribe un archivo. Uso: escribir <archivo> <contenido>
* exit Cierra el shell.
* formatear Borra todos los archivos del disco virtual.
* help Muestra la lista de comandos disponibles y su descripción.
* kill Termina un proceso por su PID. Uso: kill <pid>
* lista Lista los archivos en el disco virtual.
* listar Lista los archivos en el disco virtual.
* ls Lista los archivos en el disco virtual.
* memoria Muestra estadísticas de la memoria principal.
* memstat Muestra estadísticas de la memoria principal.
* mostrar Muestra el contenido de un archivo. Uso: mostrar <archivo>
* procesos Muestra la lista de procesos en ejecución.
* ps Muestra la lista de procesos en ejecución.
* rm Elimina un archivo del disco virtual. Uso: rm <archivo>
* run Crea y ejecuta un proceso de ejemplo. Uso: run <nombre\_proceso>
* salir Cierra el shell.
* terminar Termina un proceso por su PID. Uso: terminar <pid>
* ver Muestra el contenido de un archivo. Uso: ver <archivo>
* write Crea o sobrescribe un archivo. Uso: write <archivo> <contenido>