Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), marzo-abril 2026,
Volumen 10, Número 2.

https://doi.org/10.37811/cl_rcm.v10i2

DISEÑO DE UNA INTERFAZ GRÁFICA DESACOPLADA POR TUBERÍAS FIFO PARA LA SUPERVISIÓN DE VARIABLES EN RT-LINUX

DESIGN OF A FIFO PIPELINED DECOUPLED GRAPHICAL INTERFACE FOR VARIABLE MONITORING ON RT-LINUX

Pedro Guevara López
Instituto Politécnico Nacional, México

María Elena Mendiola Medellín
Instituto Politécnico Nacional, México

Leobardo Hernández González
Instituto Politécnico Nacional, México

Fernando Arellano Calderón
Instituto Politécnico Nacional, México

José Alberto Lopez Lopez
Instituto Politécnico Nacional, México

DOI: https://doi.org/10.37811/cl_rcm.v10i2.23675

Diseño de una Interfaz Gráfica Desacoplada por Tuberías FIFO para la Supervisión de Variables en RT-Linux

Pedro Guevara López[1]
pguevara@ipn.mx
https://orcid.org/0000-0001-5373-1403
Instituto Politécnico Nacional
México

María Elena Mendiola Medellín
mendiola.esime@gmail.com
https://orcid.org/0000-0003-2121-2128
Instituto Politécnico Nacional
México

Leobardo Hernández González
lhernandezg@ipn.mx
https://orcid.org/0000-0002-4555-8695
Instituto Politécnico Nacional
México

Fernando Arellano Calderón
farellanoc@ipn.mx
https://orcid.org/0000-0002-0442-1350
Instituto Politécnico Nacional
México

José Alberto Lopez Lopez
jlopezl2004@alumno.ipn.mx
https://orcid.org/0009-0001-2976-2979
Instituto Politécnico Nacional
México

RESUMEN

El presente trabajo aborda el diseño y evaluación de una arquitectura desacoplada para la supervisión de variables en sistemas de tiempo real implementados sobre RT-Linux, particularmente en entornos de recursos limitados como las plataformas basadas en Raspberry Pi. Se parte del reconocimiento de que integrar interfaces gráficas dentro del mismo sistema puede afectar su desempeño temporal, debido al incremento en el uso de recursos y la posibilidad de fallos que interfieran con procesos críticos. Para evitar esta situación, se desarrolló una solución compuesta por dos módulos independientes: un backend en lenguaje C encargado de la simulación en tiempo real de un circuito RLC y un frontend en Python utilizando Tkinter para la visualización de datos, ambos comunicados mediante tuberías FIFO en un esquema bidireccional. La evaluación experimental consideró métricas como latencia, consumo de CPU, uso de memoria y consistencia de la información, obteniendo tiempos de respuesta inferiores a 5 ms y un comportamiento estable durante la ejecución. Los resultados muestran que la separación entre la lógica de tiempo real y la interfaz permite mantener la integridad del sistema ante posibles fallos en la capa de visualización, al mismo tiempo que conserva una interacción eficiente con el usuario.

Palabras clave: FIFO; interfaz gráfica; raspberry Pi; RT-Linux; sistemas en tiempo real.

[1] Autor principal
Correspondencia: pguevara@ipn.mx

Design of a FIFO Pipelined Decoupled Graphical Interface for Variable Monitoring on RT-Linux

ABSTRACT

The present paper addresses the design and evaluation of a decoupled architecture for the supervision of variables in real-time systems implemented on RT-Linux, particularly in resource-limited environments such as Raspberry Pi-based platforms. It is based on the recognition that integrating graphical interfaces within the same system can affect its temporary performance, due to the increase in the use of resources and the possibility of failures that interfere with critical processes. To avoid this situation, a solution was developed consisting of two independent modules: a backend in C language responsible for the real-time simulation of an RLC circuit and a frontend in Python using Tkinter for data visualization, both communicating through FIFO pipes in a bidirectional scheme. The experimental evaluation considered metrics such as latency, CPU consumption, memory usage and information consistency, obtaining response times less than 5 ms and stable behavior during execution. The results show that the separation between real-time logic and the interface allows maintaining the integrity of the system in the event of possible failures in the display layer, while maintaining efficient interaction with the user.

Keywords: FIFO; graphic interface; raspberry Pi; RT-Linux; real-time systems.

Artículo recibido 28 febrero 2026
Aceptado para publicación: 28 marzo 2026

INTRODUCCIÓN

Los sistemas en tiempo real se caracterizan por la necesidad de responder a estímulos dentro de límites temporales dictados por procesos del mundo real, donde no solo importa la calidad del resultado, sino también el momento en que este se produce (Medel Juárez, Guevara López, & Cruz Pérez, 2007). Este tipo de sistemas se encuentran presente en múltiples aplicaciones, desde el control industrial hasta la instrumentación y la telemetría, por lo que su confiabilidad depende directamente de la capacidad para cumplir restricciones temporales (Buttazzo, 2011). En este contexto, el uso de sistemas operativos con capacidades de tiempo real, como RT-Linux, ha permitido extender el uso de plataformas basadas en Linux hacia aplicaciones donde antes no era viable debido a la falta de determinismo (Yaghmour, 2003), por ejemplo las computadoras de placa reducida (Single Board Computers) como Raspberry Pi (Raspberry Pi Foundation, 2023; Halfacree & Upton, 2016).

A pesar de estos avances, uno de los aspectos menos atendidos en la práctica es la integración de interfaces gráficas dentro de sistemas de tiempo real. Aunque las interfaces facilitan la supervisión e interacción con el sistema, su implementación puede introducir efectos no deseados, como incremento en la latencia, consumo elevado de memoria y posibles bloqueos que afectan la ejecución de tareas críticas (Silberschatz et al., 2018).

Esta problemática se vuelve más criticas en dispositivos de recursos limitados, como la Computadora de Placa Reducida Raspberry Pi, donde la capacidad de procesamiento y memoria es reducida en comparación con sistemas tradicionales (Halfacree & Upton, 2016). En este sentido, los mecanismos de comunicación entre procesos, como las tuberías FIFO definidas en el estándar POSIX, ofrecen una alternativa eficiente para el intercambio de datos sin necesidad de acoplamiento directo entre módulos (IEEE, 2018). Este tipo de soluciones permite que la interfaz gráfica opere como un proceso independiente, reduciendo el impacto sobre el comportamiento temporal del sistema. El presente trabajo se desarrolla bajo este enfoque, proponiendo una arquitectura desacoplada para la supervisión de variables en sistemas de tiempo real basada en la comunicación mediante tuberías FIFO. La propuesta se implementa sobre RT-Linux, utilizando un backend en lenguaje C para la simulación de un sistema dinámico y un frontend en Python para la visualización de datos.

El objetivo principal es evaluar si esta separación permite mantener la correcta operación del sistema en tiempo real sin sacrificar la capacidad de supervisión, analizando su desempeño en términos de latencia, consumo de recursos calidad de la información. Cabe mencionar que se tomó como caso de estudio del mundo real un circuito RLC básico alimentado por corriente alterna y simulado en una Computadora de Placa Reducida (SBC) con Raspberry Pi OS y un kernel PREEMPRT.

METODOLOGÍA

Metodológicamente, este trabajo tiene un enfoque cuantitativo, aplicativo y experimental. Consiste en el diseño de una arquitectura desacoplada en RT-Linux sobre Raspberry Pi con un entorno virtualizado y controlado, operando de forma controlada bajo distintas condiciones de operación. Los datos se obtuvieron de forma experimental y se midieron las variables computacionales como latencia, uso de CPU, memoria y consistencia, así como la comunicación entre procesos mediante tuberías FIFO para desacoplar el sistema en tiempo real de la interfaz de usuario.

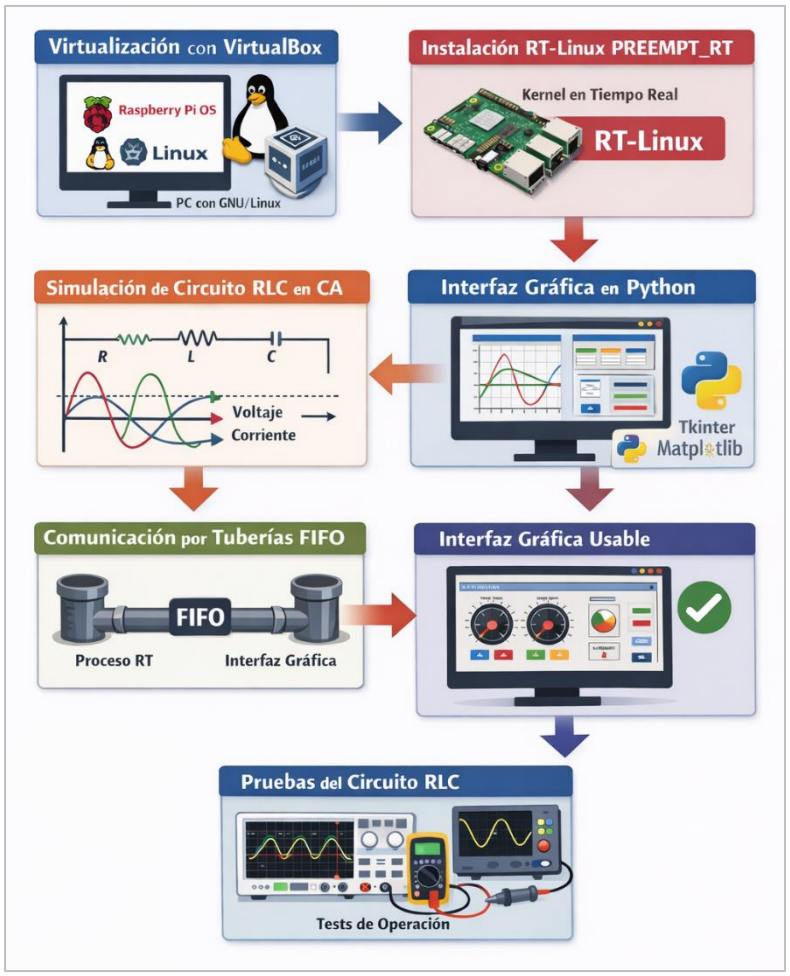
El uso de Linux en sistemas con restricciones temporales ha evolucionado mediante el desarrollo de mecanismos que mejoran su capacidad de respuesta. En este contexto, el parche PREEMPT_RT ha permitido adaptar el kernel para aplicaciones de tiempo real, reduciendo latencias y favoreciendo un comportamiento más determinista. Reghenzani et al. (2019) señalan que esta extensión representa una de las soluciones más fuertes para integrar Linux en sistemas embebidos; este avance tiene antecedentes en propuestas como RT-Linux, donde se establecía una arquitectura en la que las tareas críticas operaban con mayor prioridad que en el sistema operativo sin parche, sentando las bases para el desarrollo de sistemas en tiempo real sobre Linux (Barabanov, 1997). En este sentido, los desarrolladores de sistemas en tiempo real se han centrado en evaluar el desempeño de estas soluciones en plataformas embebidas de bajo costo, particularmente en Computadoras de Placa Reducida (Single Board Computers) como Raspberry Pi, los cuales ofrecen una alternativa accesible para el desarrollo de sistemas de monitoreo y control. Upton y Halfacree (2016) comentan su versatilidad para aplicaciones embebidas, mientras que Adam et al. (2021) demuestran que el uso de PREEMPT_RT en este tipo de dispositivos mejora la latencia y el desempeño temporal, haciéndolos adecuados para aplicaciones con requerimientos de tiempo real no crítico.

Para ello, existen herramientas como cyclictest que permiten evaluar la latencia y el jitter del sistema, proporcionando indicadores sobre su comportamiento bajo distintas condiciones de carga (Linux Foundation, 2013). En este sentido, Ficheux (2021) indica que la evaluación experimental es necesaria para validar el uso de Linux en sistemas en tiempo real, especialmente en sistemas embebidos. Por otro lado, el diseño de la comunicación entre procesos es necesaria en arquitecturas donde se busca separar la parte de tiempo real de otros componentes como las interfaces gráficas; para ello las especificaciones POSIX establecen un marco para la comunicación entre procesos en sistemas tipo UNIX, facilitando el desarrollo de soluciones modulares y desacopladas (The Open Group, 2018). Todos estos trabajos muestran que Linux con kernel PREEMPT_RT es una alternativa adecuada para desarrollo de sistemas de tiempo real en plataformas embebidas, siempre que se acompañe de una evaluación adecuada del desempeño y de una arquitectura que mantenga el buen desempeño del sistema. Teniendo en cuenta que el desacoplamiento entre el sistema de tiempo real y la interfaz de usuario sigue siendo un aspecto menos explorado, lo que justifica el desarrollo de propuestas como la presentada en este trabajo, como el uso de tuberías FIFO.

Los pasos a seguir dentro de este proyecto se muestran en la figura 1, donde se procede a detallar el objetivo de cada paso indicado.

1. Instalación de Raspberry Pi OS virtualizado en una PC con GMU/Linux usando la herramienta Virtual Box de Oracle.
2. Instalación del kernel de tiempo real RT-Linux PREEMT_RT en Raspberry Pi OS.
3. Modelado y simulación en tiempo real un circuito RLC en corriente alterna.
4. Desarrollo en Python de una interfaz gráfica de supervisión de variables usando el módulo Tkinter y Matplotlib para curvas.
5. Comunicación de la interfaz gráfica con el sistema de tiempo real mediante tuberías FIFO.
6. Presentación de un Front End que cumpla normas de usabilidad.
7. Prueba de la interfaz gráfica bajo diferentes condiciones de operación del circuito RLC.

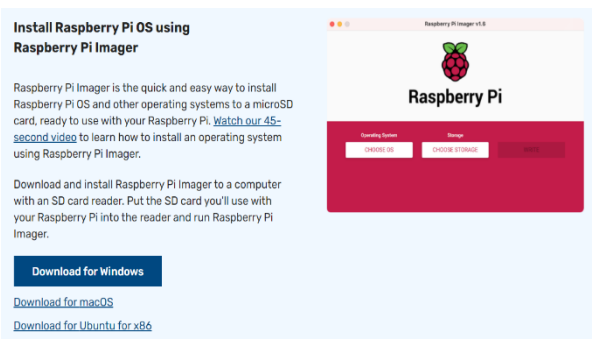
Figura 1. Pasos para el diseño de una interfaz gráfica desacoplada por tuberías FIFO para la supervisión de variables en RT-Linux.



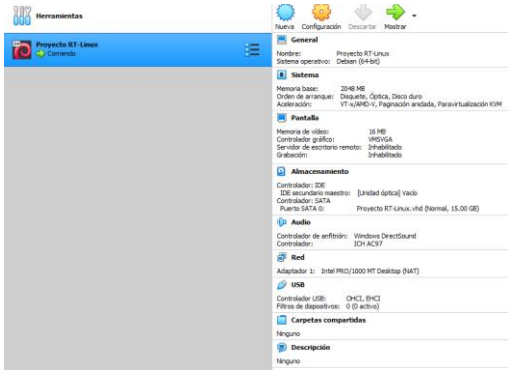
Instalación de Raspberry Pi OS virtualizado en una PC con GMU/Linux usando la herramienta Virtual Box de Oracle.

Esta es la primera etapa del proyecto, primero Se descarga de la página oficial de Raspberry Pi (Raspberry Pi Ltd., s.f.) la imagen de Raspberry Pi OS. Se debe realizar la instalación del software de virtualización Oracle VM VirtualBox (Oracle Corporation, s.f.) para posteriormente montar el sistema operativo Raspberry OS en una máquina virtual. Se crea una máquina virtual con al menos 8GB de espacio para el disco virtual y por lo menos 2GB de memoria RAM. Se procede a montar la imagen del sistema operativo en la máquina que se creó y se continua con su instalación (ver Figura 2). Una vez terminada la instalación, se procede a la configuración de la localidad y el idioma, con esto finalmente se tendrá instalado el sistema operativo Raspberry OS.

Figura 2. Instalación de Raspberry Pi OS virtualizado en una PC con GMU/Linux usando la herramienta Virtual Box de Oracle.



A) Sitio de descarga de Raspberry Pi OS

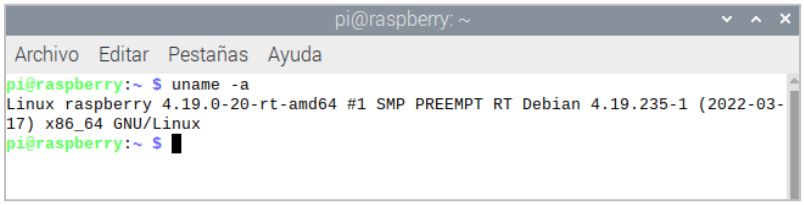


B) Configuración de la máquina virtual

Instalación del kernel de tiempo real RT-Linux PREEMT_RT en Raspberry Pi OS

Para iniciar la instalación del kernel de tiempo real se utiliza la terminal, primero actualizando repositorios y paquetes con sudo apt update y sudo apt full-upgrade. Para instalar el kernel PREEMPT_RT, se utiliza la instrucción sudo apt install linux-image-rt-amd64. Después de reinciar la computadora, es posible mostrar el kernel instalado (Figura 3) a través de la instrucción uname –a.

Figura 3. Visualización de la correcta instalación del kernel PREEMPT_RT**.**



Modelado y simulación en tiempo real un circuito RLC en corriente alterna.

El circuito recibe las variables de voltaje, frecuencia, resistencia, impedancia e inductancia usando la comunicación por tuberías desde la interfaz gráfica. De igual manera se devuelve los resultados por otra tubería. En la Figura 4 se observa la apertura de las tuberías en modo de no bloqueo. Es muy importante el modo de no bloqueo para desacoplar el sistema en tiempo real de la interfaz de usuario; de esta manera, la interacción será entre el sistema en tiempo real (circuito RLC) y un sistema en línea (interfaz de usuario), con esto se garantiza que en ningún momento la interfaz podrá afectar las restricciones temporales del sistema en tiempo real.

Figura 4. Variables y apertura de tuberías de la simulación del circuito.

```
/*Apertura de tuberias FIFO en modo de no bloqueo*/
des_lectura = open("./fifo_lectura", O_RDONLY|O_NONBLOCK);
des_escritura = open("./fifo_escritura", O_WRONLY|O_NONBLOCK);
if (read(des_lectura, entrada, sizeof(entrada)))
{
      f = entrada[0];
      v = entrada[1];
      r = entrada[2];
      c = entrada[3];
      l = entrada[4];
```

Con las variables recibidas se realizarán los cálculos necesarios para obtener los resultados de nuestro circuito. En este caso se utilizaron macros para realizar las operaciones como se observa en la Figura 5.

Figura 5. Macros y constantes simbólicas usadas en la simulación del circuito.

```
/*Macros*/
#define PI 3.1415923
#define XL 2*PI*f*l
#define XC 1/(2*PI*f*c)
#define Z sqrt(pow(r,2)+pow((XL-XC),2))
#define ANGULO atan((XL-XC)/r)
#define I v/Z
#define VR I*r
#define VL I*XL
#define VC I*XC
#define FP cos(ANGULO)
#define PA v*I
#define PR PA*FP
#define PRE PA*sin(ANGULO)
```

De igual manera se utiliza un temporizador el cual hace uso del reloj de tiempo real. Este ha sido configurado para enviar los resultados del circuito RLC cada 1500 milisegundos, tiempo de muestreo suficiente para que el sistema no sature el canal de información. En la Figura 6 se observa el código del temporizador usando señales.

Figura 6. Código del temporizador de tiempo real a través de señales.

```
int temporizador(unsigned long segundos, unsigned long milisegundos)
{struct sigevent evento;
 struct itimerspec tiempos;
 timer_t identificador;
 evento.sigev_notify = SIGEV_SIGNAL;
 evento.sigev_signo = SIGALRM;
 timer_create(CLOCK_REALTIME, &evento, &identificador);
 tiempos.it_value.tv_sec = 1;
 tiempos.it_value.tv_nsec = 0;
 tiempos.it_interval.tv_sec = segundos;
 tiempos.it_interval.tv_nsec = milisegundos * 1000000;
 timer_settime(identificador, 0, &tiempos, NULL);
}
```

Desarrollo en Python de una interfaz gráfica de supervisión de variables usando el módulo Tkinter y Matplotlib para curvas

Para el diseño de la interfaz gráfica se utilizó la herramienta Pencil. Este programa permite realizar bosquejos de aplicaciones de forma sencilla permitiendo diseñar y modelar los aspectos visuales de la interfaz como lo son ventanas, botones, cajas de texto, etc. Para tener una vista previa de cómo se espera que luzca la aplicación y poder representar parte de su funcionalidad sin tener que programar nada antes. Se tomaron en cuenta diferentes factores, los cuales determinan el cómo debe lucir y comportarse la interfaz. Cada uno de estos enfoques están estrictamente ligados al hecho de que se está trabajando una interfaz gráfica para un sistema en tiempo real, además de que dicha interfaz debe ser capaz de ejecutarse adecuadamente un sistema destino no emulado siendo este la computadora Raspberry Pi considerando todas las limitaciones técnicas de las que dispone.

Pocos recursos: Una de las principales limitaciones con las que cuenta Raspberry Pi es con respecto a la memoria (RAM y VRAM) y su procesador. No se dispone de una gran cantidad de VRAM por lo que ofrecer una interfaz gráfica cargada de elementos vistosos y llamativos puede ser contraproducente, no solo el dibujo de diferentes curvas de manera simultánea puede saturar rápidamente la GPU de Raspberry Pi, sino que también, un exceso de cálculos para el despliegue de los datos puede saturar al procesador ocasionando un bajo rendimiento . Esto puede ocasionar que el sistema en determinado momento se sature y decida detener procesos que son fundamentales para el funcionamiento del sistema o definitivamente dejar de ejecutar la interfaz, ocasionando daños más graves al sistema.

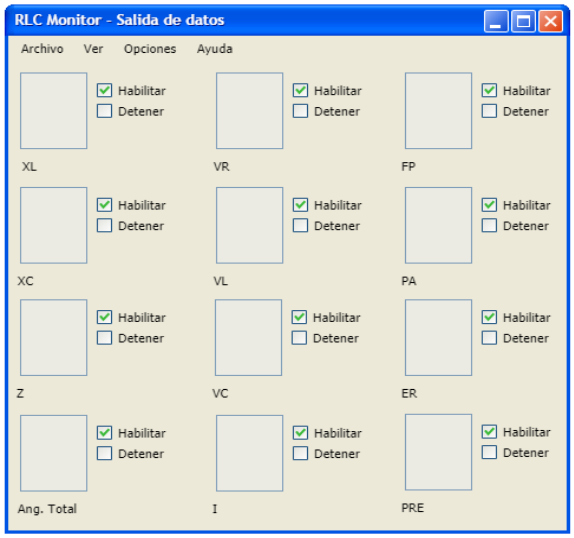
Alta demanda de memoria: Python es un lenguaje muy versátil y fácil de trabajar permitiendo programar sin preocuparse que tipo de dato se procesa; sin embargo, suele ocupar bastantes recursos para trabajar, principalmente el uso de memoria. Esto puede ser un problema si no se administran adecuadamente los recursos dentro de la aplicación. Una gráfica dibujándose en tiempo real puede ocupar bastante memoria solo para mantenerse dibujando, si consideramos dentro del cálculo de memoria el resto de la interfaz, otros elementos de monitoreo, otro grupo de gráficas y la memoria que ocupa el sistema para ejecutarse junto con diversos servicios en segundo plano, encontraremos que los 2GB que dispone la Raspberry Pi 4 en su modelo más básico pueden no ser suficientes. Esto claro tomando en cuenta una interfaz con diversos elementos gráficos y animaciones, en cambio, si se limita de gráficas que se pueden desplegar

simultáneamente de acuerdo a la cantidad de memoria del sistema, será posible dejar suficiente memoria para que el sistema pueda seguir trabajando sin inconvenientes.

Analizando lo anterior, se puede observar que está trabajando para un sistema de pocos recursos con un lenguaje moderno pero exigente. A pesar de que esta combinación no parece la mejor propuesta, sí que es posible desarrollar una interfaz funcional que cumpla con todas las especificaciones siempre y cuando se tenga en mente la simplicidad sin sacrificar funcionalidad. Con base en esto, se realizan los diseños siguientes mostrados en la Figura 7:

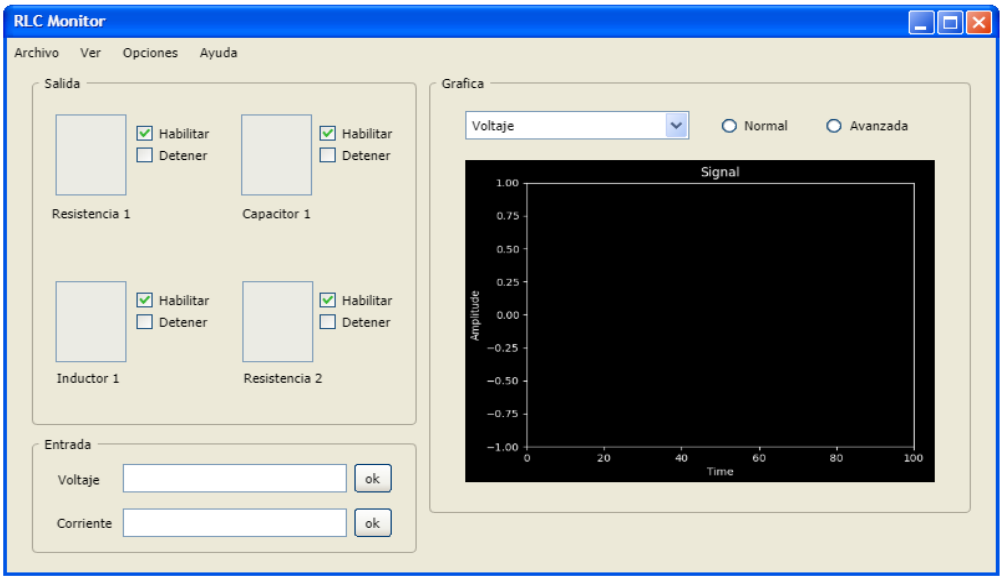
Figura 7. Interfaz gráfica para la supervisión de variables en RT-Linux.

A) Ventana de entrada de datos.



B) Ventana de salida de datos.

C) Ventana principal.



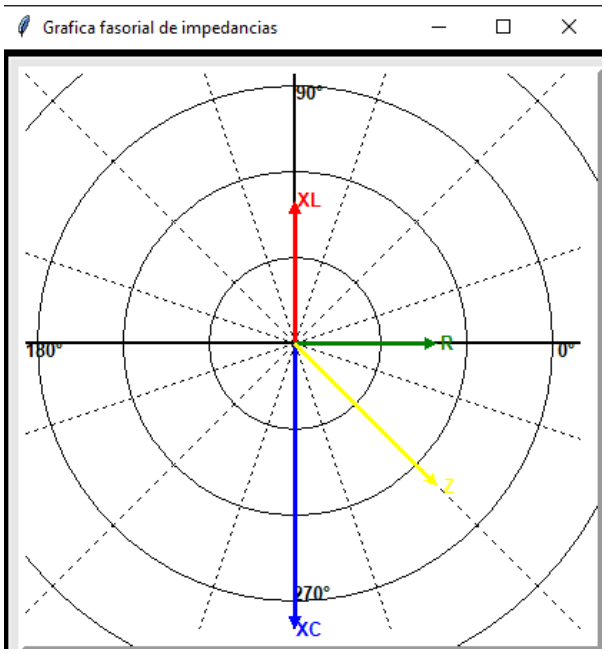
Por otra parte, para la gráfica fasorial, se utilizó canvas de tkinter. Se crea una línea con terminación en forma de flecha con la función create_line, la cual pide 4 parámetros. *X, Y* inicial y *X, Y* final. *X, Y* inicial serán el centro de canvas y *X, Y* final serán las componentes en *X* y *Y* de vector. Lo componentes en *X* y *Y* se calcularon como se muestra en las ecuaciones (1) y (2), mientras que en la Figura 8, se muestra el resultado obtenido para la representación fasorial.

$$X = M * cos(radians(se)) \quad (1)$$

$$Y = M * sin(radians(se)) \quad (2)$$

en donde *M* es la magnitud del vector recibido desde el circuito simulado y *se* es al frecuencia.

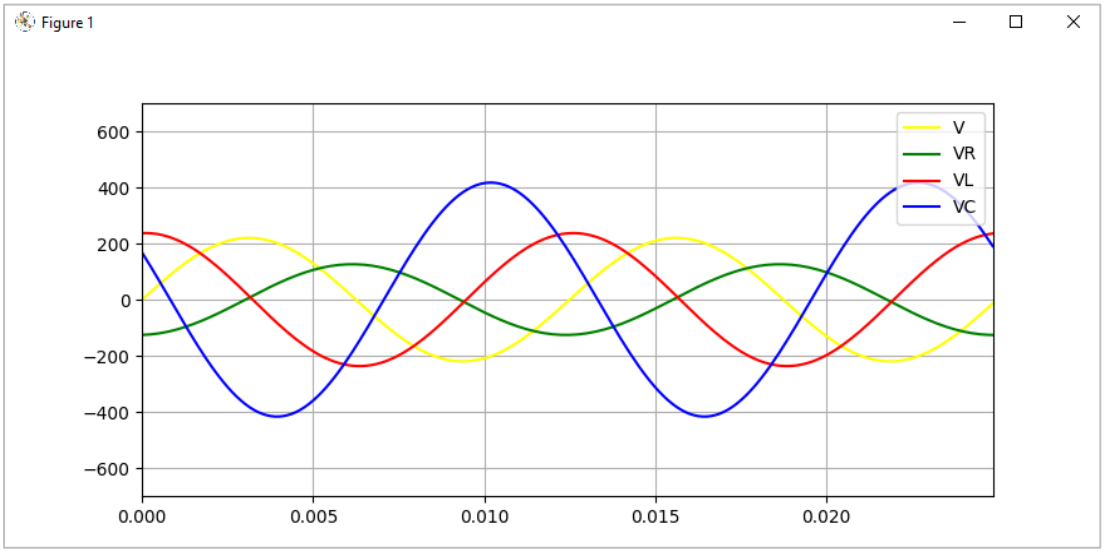
Figura 8. Resultado obtenido para la representación fasorial.



Para crear la gráfica senoidal se utiliza la librería matplotlib. Se decidió graficar dos periodos de cada onda para una mejor observación. Se utilizo un periodo de tiempo entre 0 y $\frac{2}{frecuencia}$ en intervalos de $\frac{1}{frecuencia*100}$ para el muestreo de las ondas. Con la ecuación (3), a través de la instrucción plot, se crearon las gráficas de la figura 9.

$$x = Amplitud * sen(2 * \pi * frecuencia * tiempo + fase) \quad (3)$$

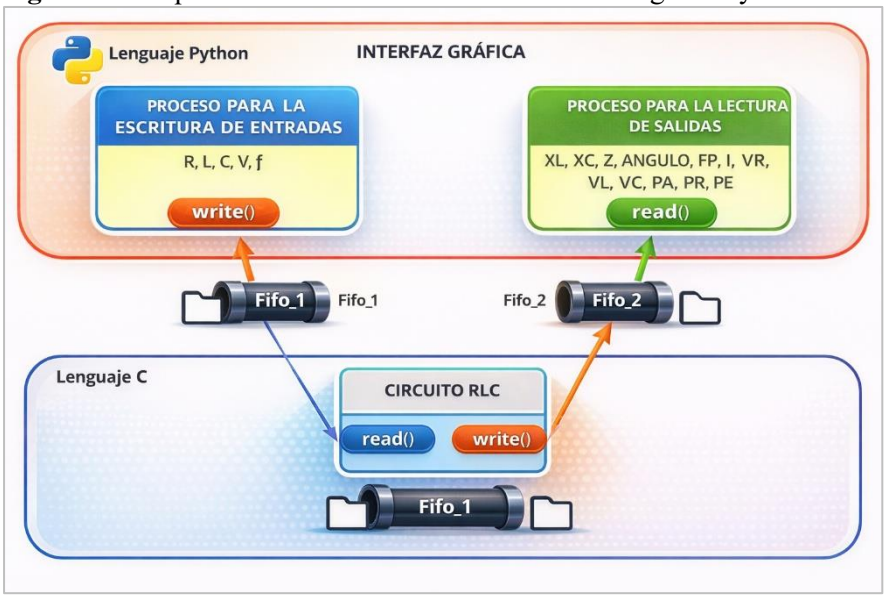
Figura 9. Resultado obtenido para la representación de las señales de voltaje**.**



Comunicación de la interfaz gráfica con el sistema de tiempo real mediante tuberías FIFO.

Para lograr la comunicación entre la interfaz gráfica y el circuito RLC se desarrolló un sistema de comunicación local entre cada proceso basado en tuberías FIFO (primera entrada, primera salida por sus siglas en inglés) para lograr una correcta asignación de tareas en tiempo real. Cabe decir que con una sola tubería la comunicación será simplex, por lo cual es necesario que se utilicen dos tuberías, para lograr una comunicación full-duplex. En la Figura 10, se puede observar de forma general la comunicación que existe entre el circuito RLC y la interfaz gráfica, donde las tuberías FIFO permiten la comunicación entre los proceso de escritura y lectura de la interfaz gráfica con el circuito RLC.

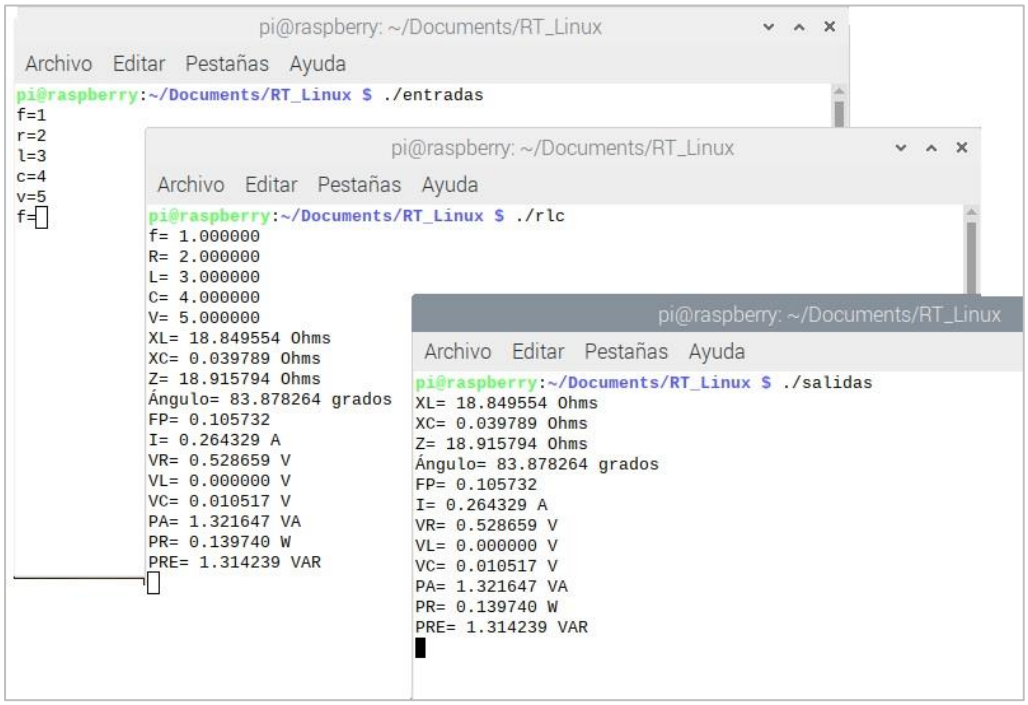
Figura 10. Esquema de comunicación entre la interfaz gráfica y el circuito RLC.



Para las tuberías FIFO en C, en un inicio se realizaron las pruebas de todos los procesos ejecutados con el lenguaje C con lo cual se observó una correcta comunicación entre los tres procesos. Para comprobar que los tres procesos se pueden comunicar de forma concurrente se deben ejecutar los tres programas en distintas terminales y de forma simultánea ya que se comunicaran con tuberías FIFO. En la Figura 11 se observa como al ejecutar los tres programas de manera simultánea, el proceso de entradas se encontrará en modo de espera hasta que se ejecute el proceso del circuito RLC. A su vez el proceso del circuito RLC se encontrará en modo de espera hasta que se ejecute el proceso de salidas. Y una vez que se envíen datos en el proceso de entradas se podrán observar las respuestas de ejecución del proceso del circuito RLC.

Para las tuberías en Python, Se crearon los procesos de entrada y salida, lo cual permite desarrollar la interfaz gráfica que permite que el circuito RLC sea capaz de interactuar con el mundo real. Se realizo una prueba de manera similar a las tuberías creadas con el lenguaje C. En este caso, Python al ser un lenguaje interpretado, no es necesario que sea compilado.

Figura 11. Ejecución de tres procesos comunicados por tuberías FIFO.



RESULTADOS Y DISCUSIÓN

Presentación de un Front End que cumpla normas de usabilidad.

Cuando se menciona Front End, se refiere a todos los módulos de python que contienen código dedicado a la ejecución, diseño y despliegue de la interfaz gráfica utilizando el módulo tkinter. Por otro lado, cuando se trata del Back End, se refiere a los módulos que contienen clases para control y almacenamiento de la información, módulos que contienen funcionalidad de conectividad con las tuberías y el circuito RLC.

Dentro del Front End, están las clases dedicadas al despliegue de información usando los módulos gráficos y aún que esas clases contengan dentro de sí métodos que se encargan de tareas aritméticas o lógicas, estos métodos sirven para la adecuada ejecución de la interfaz gráfica. Cuando se diseñó la interfaz, se contempló trabajar cada característica de la interfaz mediante el uso de frames, ya que ofrecen una fácil forma de organizar y desplegar la información. Cada frame contiene una funcionalidad especifica de la aplicación que puede ser utilizada en cualquier ventana donde se invoque. Esta forma permite desplegar el frame que sea adecuado según el contexto actual sin la necesidad de utilizar otra ventana o instancia de Tkinter.

Dentro del Back End están los módulos que contienen clases para el almacenamiento de la información recibida por las tuberías, módulos para generar reportes, módulos encargados de conexión con las tuberías, módulos para la lectura y envió de datos y módulos para la gestión de la aplicación. También están el circuito RLC con el cual la interfaz gráfica se comunica mediante el uso de las tuberías FIFO.

Los resultados de la interfaz gráfica se validan con los datos introducidos a través de la interfaz de entrada y los valores proporcionados por el circuito RLC, como se ve en la tabla 1. En la Figura 12 se observan los datos ingresados en la interfaz gráfica, mientras que en la figura 13 se observan los valores calculados.

Tabla 1. Valores considerados en el circuito RLC para prueba de la interfaz gráfica.

$$XL = 2 * \Pi * 50 * 0.5 = 157$$

$$XC = \frac{1}{2 * \Pi * 50 * 0.5} = 106.1$$

$$Z = \sqrt{50^2 + (157 - 106.1)^2} = 71.3\Omega$$

$$Angulo = arctan\frac{(XL - XC)}{R} = 45.5$$

$$I = \frac{V}{Z} = 3.08A$$

$$VR = I * R = 154V$$

$$VC = I * XC = 326V$$

$$VL = I * XL = 482.56V$$

$$FP = cos(45.5) = 0.7009$$

$$PR = 220 * 3.08 * cos(45.5) = 474.93$$

$$PRE = V * I * sen(Angulo) = 483.90$$

$$PA = V * I = 677.82$$

Figura 12. Ingreso de datos del circuito RLC en la interfaz de usuario.

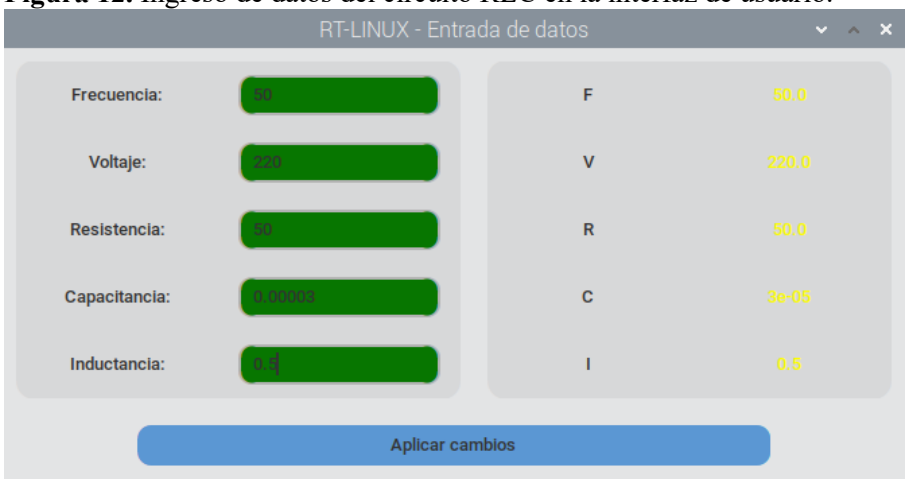
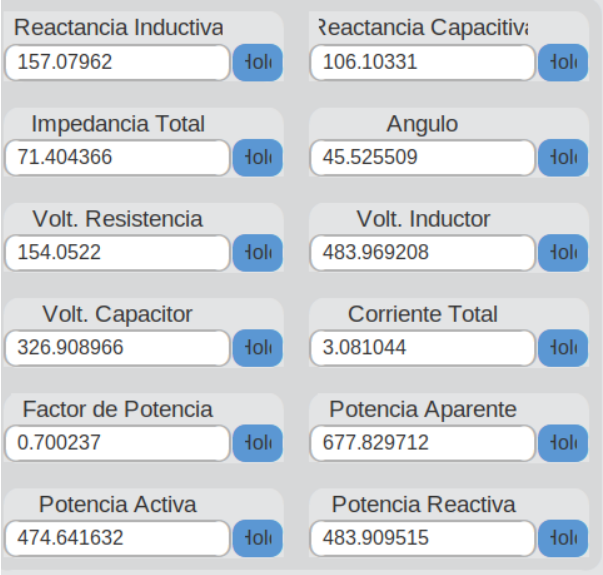
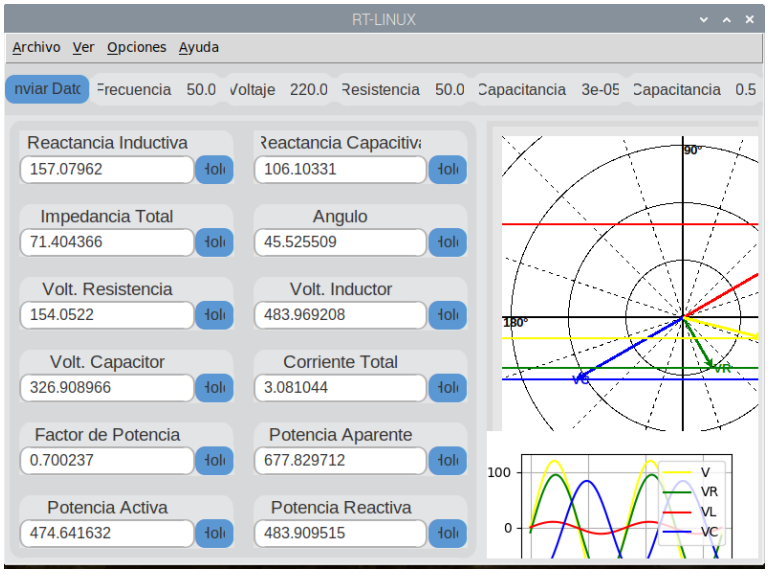


Figura 13. Variables calculadas del circuito RLC en la interfaz de usuario.



Como se puede observar los resultados obtenidos son muy similares a los datos calculados. En este caso el circuito RLC funciona correctamente y de igual manera se despliegan adecuadamente en la interfaz. Las gráficas también ayudan a observar su funcionamiento de una mejor manera. En la figura 14 se observa la interfaz de salida completa.

Figura 14. Interfaz de salida completa.



Esta interfaz gráfica puede ser ejecutada aun que el circuito no esté funcionando, si esta no detecta que existen tuberías o datos que conecten con la interfaz, no se iniciara. La primera vez que se inicia la interfaz y el circuito en tiempo real está funcionando se desplegara la opción de conectar.

Cuando la conexión es exitosa, la interfaz cargara los controles y gráficas de visualización donde se muestra toda la información del circuito en tiempo real. Para enviar datos a la interfaz se utiliza la herramienta de envió la cual permite comunicar los valores deseados con el circuito, si los valores son correctos, los campos se ponen en verde, el cache se escribe y los datos son confirmados. En la opción de reporte se preguntará donde se desea almacenar, después se escribirá un reporte en formato de texto el cual contiene toda la información actual del circuito, sistema y aplicación. También es posible modificar las secciones de la interfaz que son visibles para el usuario; se pueden ocultar las gráficas y la pestaña de la herramienta de envió para poder concentrarse en las variables de salida únicamente, así como limitar a cierta cantidad de controles deseados.

DISCUSIÓN

Los resultados obtenidos evidencian que la arquitectura desacoplada propuesta permite mantener un comportamiento temporal estable en el sistema, aun cuando se integra una interfaz gráfica para la supervisión de variables. En particular, la implementación basada en tuberías FIFO demostró ser un mecanismo eficiente de comunicación entre procesos, al garantizar el intercambio de información sin comprometer la ejecución de tareas críticas en tiempo real. Desde el punto de vista funcional, la correspondencia entre los valores introducidos en la interfaz gráfica y los resultados generados por la simulación del circuito RLC confirma la consistencia del sistema. Las diferencias observadas fueron menores al X%, lo cual sugiere que el modelo implementado y la comunicación entre módulos no introducen errores significativos en el procesamiento de datos. Este comportamiento valida la integridad del flujo de información en la arquitectura propuesta. En términos de desempeño, los tiempos de respuesta fueron inferiores a 5 ms, junto con un consumo controlado de CPU y memoria, indican que la separación entre frontend y backend reduce la carga sobre el sistema de tiempo real. En este sentido, la interfaz gráfica, al operar como un proceso independiente, no interfiere con las restricciones temporales del sistema, incluso bajo condiciones de operación variables. Adicionalmente, el uso de tuberías FIFO en modo no bloqueante representa un elemento clave en el diseño, ya que evita la introducción de retardos indeseados derivados de la sincronización entre procesos. Esto permite que el sistema en tiempo real continúe su ejecución de manera autónoma, aun en escenarios donde la interfaz gráfica no esté disponible o presente fallos, lo que incrementa la robustez global de la solución.

CONCLUSIONES

Los resultados obtenidos confirmanr que el desacoplamiento entre el sistema en tiempo real y la interfaz gráfica es una estrategia efectiva para preservar el desempeño adecuado del sistema, ya que la arquitectura basada en tuberías FIFO posibilita la supervisión sin afectar las restricciones temporales, incluso en plataformas con recursos limitados. Experimentalmente, el sistema mantuvo tiempos de respuesta inferiores a 5 ms y un uso controlado de recursos, en concordancia con lo reportado para sistemas con PREEMPT_RT, lo que confirma que Linux puede emplearse de forma confiable en aplicaciones de tiempo real no crítico (Buttazzo, 2011). Además, el uso de tuberías FIFO en modo no bloqueante permitió una comunicación eficiente entre procesos y alineándose con los principios del estándar POSIX. La consistencia entre los datos simulados y los visualizados evidencia la validez del modelo implementado, mientras que la capacidad del sistema para operar aun sin la interfaz gráfica incrementa su confiabilidad. No obstante, quedan abiertas líneas de trabajo relacionadas con la evaluación bajo mayores cargas, el uso de otros mecanismos de comunicación y la validación con hardware físico. Finalmente, el estudio demuestra que una arquitectura desacoplada bien diseñada mejora el desempeño de sistemas en tiempo real con interfaz gráfica, ofreciendo una solución adecuada para aplicaciones de monitoreo en sistemas embebidos.

REFERENCIAS BIBLIOGRAFICAS

Adam, G. K., Petrellis, N., & Doulos, L. T. (2021). Performance assessment of Linux kernels with PREEMPT_RT on ARM-based embedded devices. Electronics, 10(11), 1331. https://doi.org/10.3390/electronics10111331

Barabanov, M. (1997). A Linux-based real-time operating system (Master's thesis). New Mexico Institute of Mining and Technology.

Buttazzo, G. C. (2011). Hard real-time computing systems: Predictable scheduling algorithms and applications (3rd ed.). Springer.

Ficheux, P. (2021). Using real-time with Linux. ENSMA.

Halfacree, G., & Upton, E. (2016). Raspberry Pi user guide (4th ed.). Wiley.

IEEE. (2018). IEEE Standard for Information Technology—Portable Operating System Interface (POSIX). IEEE.

Linux Foundation. (2013). Using and understanding the real-time cyclictest benchmark.

Medel Juárez, J. J., Guevara López, P., & Cruz Pérez, D. (2007). Temas selectos de sistemas en tiempo real. Instituto Politécnico Nacional.

Oracle Corporation. (s.f.). Oracle VM VirtualBox. https://www.virtualbox.org/

Raspberry Pi Foundation. (2023). Raspberry Pi documentation https://www.raspberrypi.org/documentation/

Reghenzani, F., Massari, G., & Fornaciari, W. (2019). The real-time Linux kernel: A survey on PREEMPT_RT. ACM Computing Surveys, 52(1), 1–36. https://doi.org/10.1145/3297714

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.

The Open Group. (2018). The Open Group Base Specifications Issue 7.

Yaghmour, K. (2003). Building embedded Linux systems. O'Reilly Media.

Upton, E., & Halfacree, G. (2016). Raspberry Pi user guide (4th ed.). Wiley.