Proyecto Final: Centro Multimedia

Marco Antonio Montelongo Alquicira

1. Introducción

Un sistema embebido integra hardware de cómputo junto con software embebido como uno de sus componentes más importantes. Es un sistema computacional dedicado a aplicaciones o productos específicos. Puede ser un sistema independiente o parte de uno mayor. Dado que usualmente su software está embebido en memoria ROM (*Read Only Memory*), no necesita memoria secundaria como una computadora. La complejidad de los sistemas embebidos puede variar significativamente según la tarea para la que estén diseñados, abarcando desde un microcontrolador simple hasta un conjunto de chips con periféricos y redes conectadas [1].

La Raspberry Pi 4 Model B (Pi4B) es una microcomputadora compacta y de alto rendimiento que mejora significativamente las capacidades de CPU, GPU y entrada/salida respecto a su predecesora. Está disponible con 1, 2, 4 u 8 GB de memoria LPDDR4, incluye un procesador ARM Cortex-A72 de 64 bits a 1.5 GHz y soporte para video 4Kp60 en salidas duales HDMI. Además, cuenta con conectividad WiFi, Bluetooth 5.0, puertos USB 2.0 y 3.0, y hasta 28 pines GPIO configurables. Gracias a su diseño versátil y soporte para Linux, es ideal para proyectos educativos, industriales y multimedia [2].

Un *media center*, o centro multimedia, es un software que permite gestionar y disfrutar contenido multimedia. Puede reproducir música, películas, mostrar imágenes en galerías, entre otros. Todo desde contenido multimedia almacenado en un disco duro local o accediendo a este contenido a través de la red. Estos centros multimedia pueden ejecutarse en dispositivos móviles, computadoras personales (*PC*), computadoras de placa única (*Single Board Computers, SBC*) como es el caso de OSMC sobre Raspberry Pi, en televisores inteligentes (*smart TV*), entre otros [3].

1.1. Objetivo

El objetivo de este proyecto es desarrollar un sistema embebido de entretenimiento que permita a los usuarios reproducir películas, videos, música y fotografías tanto desde medios extraíbles como mediante servicios en línea seleccionados. Este sistema está diseñado para ofrecer una experiencia intuitiva y fluida, garantizando compatibilidad con contenido multimedia moderno y priorizando una interfaz accesible y fácil de usar.

2. Lista de materiales

- Raspberry Pi 4.
- Fuente de alimentación para la Raspberry Pi.
- Cable micro HDMI a VGA o adaptador micro HDMI a HDMI.
- Tarjeta microSD de 8 GB (mínimo).
- Unidad USB con contenido multimedia.
- Monitor.
- Teclado.

- Mouse.
- Bocinas o audífonos con conector de 3.5 mm.

3. Descripción del funcionamiento de los componentes

3.1. Cable micro HDMI a VGA

- **Transmisión de video:** Este cable permite conectar la Raspberry Pi a un monitor con entrada VGA, facilitando la salida de video en alta calidad. Es una solución práctica cuando el monitor no dispone de entrada HDMI.

3.2. Unidad USB

- **Almacenamiento de archivos:** La unidad USB emplea tecnología de memoria no volátil, lo que asegura la conservación de los datos sin necesidad de energía constante. Es ideal para guardar y transferir contenido multimedia.
- **Conexión rápida:** La interfaz USB permite transferir datos de manera ágil y eficiente entre la Raspberry Pi y el dispositivo de almacenamiento externo.

3.3. Teclado y Mouse

- **Control e interacción:** Estos periféricos sirven para la navegación por la interfaz del sistema y el control de sus funciones. Su conexión se realiza mediante los puertos USB de la Raspberry Pi, asegurando una respuesta inmediata.

3.4. Monitor

- **Pantalla de salida:** Es el dispositivo encargado de mostrar las imágenes y videos procesados por la Raspberry Pi. Su función es fundamental para la interacción visual con el sistema.
- **Compatibilidad con adaptadores:** Puede conectarse mediante cables HDMI o adaptadores micro HDMI a VGA para garantizar compatibilidad con diferentes modelos y tecnologías de monitores.

3.5. Fuente de Alimentación

- **Energía estable:** Proporciona el suministro eléctrico necesario para el correcto funcionamiento de la Raspberry Pi. Se recomienda una fuente de alimentación original de 5V y al menos 3A para evitar fallos de energía y garantizar la estabilidad del sistema.

3.6. Bocinas y Audífonos

- **Salida de audio:** Este componente se conecta mediante un conector de audio de 3.5 mm para proporcionar sonido claro y de calidad en la reproducción multimedia. Es un elemento esencial para disfrutar plenamente de la experiencia auditiva del sistema.

3.7. Tarjeta microSD

- **Sistema de arranque:** Actúa como el disco principal donde se almacena el sistema operativo (Raspbian Lite) y los archivos fundamentales del proyecto. Permite el inicio del sistema y la ejecución de las aplicaciones multimedia.

4. Información sobre el cuidado de la salud

Para garantizar la seguridad del usuario, evite el contacto directo con la Raspberry Pi cuando está en funcionamiento, ya que algunos componentes pueden calentarse y causar molestias leves. Asegúrese de trabajar en un espacio bien iluminado y ventilado para reducir la fatiga visual y prevenir molestias respiratorias en caso de acumulación de calor. También es recomendable desconectar el dispositivo de la corriente antes de manipularlo para evitar riesgos de descargas eléctricas.

5. Información sobre el cuidado de los componentes

Es importante almacenar los componentes electrónicos en un lugar seco y libre de polvo para evitar la humedad o partículas pequeñas. Al conectar o desconectar dispositivos como la Raspberry Pi o el monitor, hágalo siempre con el sistema apagado para prevenir cortocircuitos. Además, asegúrese de manejar la Raspberry Pi utilizando una superficie antiestática para evitar daños por electricidad estática, y evite colocar objetos pesados sobre los cables o la placa, ya que podrían doblarse o dañarse.

6. Configuración de la tarjeta

Para el correcto funcionamiento de la Raspberry Pi, es necesario conectar todos los periféricos requeridos, como el teclado, el mouse y el monitor, asegurándose de utilizar los cables correspondientes. Una vez que los periféricos estén listos, encienda la Raspberry Pi conectándola a una fuente de alimentación.

Es indispensable preparar el sistema operativo para la Raspberry Pi. Se debe descargar e instalar *Raspberry Pi OS Lite* en una tarjeta microSD; esta debe cumplir con las especificaciones necesarias para ejecutar el sistema multimedia. Además, es necesario instalar las dependencias requeridas para el correcto funcionamiento del sistema, como:

1. **tkinter**[4]: Es un módulo de Python que proporciona herramientas para desarrollar interfaces gráficas de usuario (GUI). Viene integrado con Python pero puede necesitar instalación adicional.
   - Instalación: `sudo apt install python3-tk`
2. **Pillow (PIL)**[5]: Es una biblioteca de Python para la manipulación y procesamiento de imágenes. Es una versión mejorada de PIL (Python Imaging Library).
   - Instalación: `sudo apt install python3-pil`
3. **screeninfo**[6]: Permite obtener información sobre las pantallas conectadas al sistema, como resoluciones y configuraciones.
   - Instalación: `sudo apt install python3-screeninfo`
4. **pyudev**[7]: Biblioteca Python para interactuar con `udev`, que permite gestionar dispositivos conectados al sistema.
   - Instalación: `sudo apt install python3-pyudev`
5. **chromium**[8]: Es un navegador web de código abierto ampliamente utilizado.
   - Instalación: `sudo apt install chromium`
6. **pulseaudio**[9]: Es un servidor de sonido utilizado en sistemas Linux para la gestión avanzada de audio.
   - Instalación: `sudo apt install pulseaudio`
7. **vlc y python3-vlc**[10]: VLC es un reproductor multimedia versátil, y `python3-vlc` proporciona una interfaz para controlar VLC desde Python.
   - Instalación: `sudo apt install vlc python3-vlc`

8. **libwidevinecdm0**[11]: Biblioteca necesaria para la reproducción de contenido DRM (Digital Rights Management), como el de servicios de streaming.
    - Instalación: `sudo apt install libwidevinecdm0`

Con las dependencias instaladas, la Raspberry Pi estará lista para su configuración y personalización como un centro multimedia completo.

7. Desarrollo de los módulos

7.1. main.py

El siguiente código implementa una clase llamada `CentroMultimedia`, que crea una aplicación gráfica de un centro multimedia interactivo. Este centro permite navegar entre diferentes interfaces (Red, USB y Streaming) o apagar el sistema. Se utilizan bibliotecas como `tkinter` para la interfaz gráfica, `Pillow` para la manipulación de imágenes, `os` para ejecutar comandos del sistema y otras clases externas para manejar funciones específicas como USB, Streaming y Red. A continuación, se detalla el propósito general del código y de cada función.

1. **Clase** `CentroMultimedia`:
    - Encapsula la lógica para un menú principal que permite acceder a tres funcionalidades principales: interfaz de Red, interfaz USB e interfaz de Streaming.
    - Integra elementos dinámicos como un fondo, barras superior e inferior, y botones para navegar entre las diferentes funcionalidades.
2. **Método** `__init__`:
    - Inicializa los parámetros principales, incluyendo las dimensiones dinámicas de la pantalla obtenidas mediante `screeninfo`.
    - Configura la ventana raíz (`root`) para que abarque toda la pantalla y llama al método `mostrar_menu_principal`.
3. **Método** `limpiar_ventana`:
    - Elimina todos los elementos actuales de la ventana, preparando la interfaz para ser reutilizada o actualizada.
4. **Método** `salir_sistema`:
    - Ejecuta el comando `shutdown` del sistema operativo para apagar el dispositivo.
5. **Método** `mostrar_menu_principal`:
    - Crea la interfaz principal del centro multimedia, incluyendo un fondo dinámico, barras de encabezado y pie, y botones para acceder a las diferentes funcionalidades o apagar el sistema.
    - Los botones están organizados dinámicamente según las dimensiones de la pantalla.
6. **Método** `mostrar_usb`:
    - Cambia la interfaz actual a la destinada para manejar dispositivos USB.
    - Llama a la clase externa `USBInterface` para mostrar la funcionalidad de USB.
7. **Método** `mostrar_streaming`:

- Cambia la interfaz actual a la destinada para manejar funcionalidades de streaming.
- Llama a la clase externa `StreamingInterface` para mostrar la funcionalidad de streaming.

8. **Método** `mostrar_red`:
   - Cambia la interfaz actual a la destinada para manejar funciones relacionadas con redes.
   - Llama a la clase externa `RedInterface` para mostrar la funcionalidad de red.

7.2. audio_usb.py

El siguiente código implementa una clase llamada `AudioUSB`, que representa un reproductor de música con interfaz gráfica desarrollado en Python. Utiliza `tkinter` para la GUI, `Pillow` para manejar imágenes, `vlc` para la reproducción de música, y `screeninfo` para adaptar la interfaz a las dimensiones de la pantalla. A continuación, se describen sus funcionalidades generales y los métodos principales.

1. **Clase** `AudioUSB`:
   - Encapsula la lógica de un reproductor de música, incluyendo la interfaz gráfica y la gestión de canciones.
   - Permite funciones como reproducir, pausar, cambiar canciones y regresar a un menú anterior.
2. **Método** `__init__`:
   - Inicializa los atributos de la clase, como la lista de canciones, el índice de la canción actual y el reproductor de VLC.
   - Llama al método `mostrar_interfaz_audio()` para configurar la interfaz gráfica.
3. **Método** `limpiar_frame`:
   - Elimina la interfaz gráfica actual y detiene cualquier reproducción activa.
   - Se utiliza para reiniciar la ventana o cambiar entre vistas.
4. **Método** `mostrar_interfaz_audio`:
   - Configura dinámicamente la interfaz gráfica del reproductor de música.
   - Incluye un fondo, un encabezado, un pie de página, una imagen representativa y botones de control.
5. **Método** `crear_botones`:
   - Genera botones interactivos para controlar la reproducción (anterior, pausar, reanudar, siguiente, volver).
   - Ajusta su posición y tamaño según las dimensiones de la pantalla.
6. **Método** `reproducir_cancion`:
   - Configura y comienza la reproducción de la canción actual.
   - Actualiza la interfaz para mostrar el título de la canción en reproducción.
7. **Método** `iniciar_temporizador`:

- Configura un temporizador basado en la duración de la canción actual para pasar automáticamente a la siguiente.

8. **Métodos** `pausar_temporizador` **y** `reanudar_temporizador`:
   - `pausar_temporizador`: Detiene el temporizador activo.
   - `reanudar_temporizador`: Restaura el temporizador pausado para continuar con el cambio de canción.
9. **Método** `siguiente_cancion`:
   - Avanza a la siguiente canción de la lista y reinicia la reproducción.
10. **Método** `cancion_anterior`:
    - Retrocede a la canción anterior en la lista y reinicia la reproducción.
11. **Métodos** `pausar_cancion` **y** `reanudar_cancion`:
    - `pausar_cancion`: Pausa la reproducción actual y ajusta el temporizador.
    - `reanudar_cancion`: Reanuda la reproducción desde el punto donde se pausó.
12. **Método** `volver`:
    - Limpia la ventana actual y llama al *callback* definido para regresar a la interfaz anterior.

7.3. usb.py

El siguiente código implementa una clase llamada `USBInterface`, diseñada para gestionar una interfaz gráfica que permite interactuar con dispositivos USB. La clase detecta archivos multimedia (imágenes, audio y video), actualiza la interfaz dinámicamente según los dispositivos conectados y habilita funcionalidades específicas para manejar estos archivos. A continuación, se detallan las funcionalidades generales del código y sus métodos principales.

1. **Clase** `USBInterface`:
   - Gestiona la detección y manejo de dispositivos USB.
   - Integra elementos gráficos para mostrar y seleccionar diferentes tipos de archivos multimedia (fotos, videos, música).
   - Utiliza un sistema de monitoreo en un hilo separado para detectar y actualizar dispositivos conectados.
2. **Método** `__init__`:
   - Inicializa los elementos principales de la clase, como la interfaz gráfica, el administrador de medios (`MediaManager`) y una cola (`Queue`) para comunicarse con el hilo de monitoreo.
   - Comienza un hilo que ejecuta el monitoreo continuo de dispositivos USB.
3. **Método** `limpiar_frame`:
   - Elimina el contenido del marco actual de la interfaz gráfica, preparando el espacio para actualizar o cambiar de vista.
4. **Método** `mostrar_interfaz_usb`:

- Configura la interfaz gráfica principal para la interacción con dispositivos USB.
- Incluye elementos como un fondo, una barra superior, un botón para regresar al menú y botones para las funcionalidades de fotos, música y videos.

5. **Método** `crear_botones_usb`:
   - Genera dinámicamente botones para manejar imágenes, música y videos detectados en dispositivos USB.
   - Los botones están inicialmente deshabilitados y se activan cuando se detectan los archivos correspondientes.
6. **Método** `accion_boton`:
   - Ejecuta la acción correspondiente al botón seleccionado (visualizar fotos, reproducir música o videos).
   - Abre una nueva ventana para manejar el contenido multimedia seleccionado.
7. **Método** `actualizar_interfaz_usb`:
   - Monitorea los cambios en la cola (`Queue`) para detectar nuevos dispositivos USB conectados o desconectados.
   - Habilita o deshabilita los botones de la interfaz según el contenido detectado en el dispositivo.
   - Se ejecuta periódicamente para mantener la interfaz actualizada.

7.4. red.py

El siguiente código implementa una clase llamada `RedInterface`, que gestiona una interfaz gráfica para configurar y conectarse a redes WiFi. Utiliza `tkinter` para la interfaz, `screeninfo` para obtener dimensiones de la pantalla y una clase externa llamada `WifiManager` para manejar operaciones relacionadas con la red WiFi. A continuación, se describen los aspectos generales del código y sus principales métodos.

1. **Clase** `RedInterface`:
   - Proporciona una interfaz gráfica para listar redes WiFi, conectarse a una red seleccionada, y mostrar el estado actual de la conexión.
   - Permite al usuario interactuar con las redes disponibles mediante botones y una entrada para la contraseña.
2. **Método** `__init__`:
   - Inicializa la clase con atributos básicos, como las redes detectadas, la red seleccionada y el estado de conexión.
   - Llama a `obtener_estado_inicial` para preparar la lista de redes disponibles y determinar si el dispositivo está conectado.
3. **Método** `obtener_estado_inicial`:
   - Consulta el estado de conexión actual mediante `WifiManager`.
   - Escanea las redes WiFi disponibles y elimina duplicados para generar una lista única.

4. **Método** `mostrar_interfaz_red`:
    - Configura la interfaz gráfica con elementos como un fondo, botones para regresar o refrescar redes, y una lista de redes disponibles.
    - Incluye secciones para mostrar el estado de conexión y permitir la entrada de contraseñas.
5. **Método** `crear_botones_red`:
    - Genera dinámicamente botones para cada red disponible, permitiendo seleccionarlas.
    - Incluye botones de navegación para desplazarse entre redes si la lista es extensa.
6. **Métodos** `subir_redes` **y** `bajar_redes`:
    - `subir_redes`: Desplaza hacia arriba la lista de redes mostradas.
    - `bajar_redes`: Desplaza hacia abajo la lista de redes mostradas.
7. **Método** `seleccionar_red`:
    - Permite seleccionar una red WiFi de la lista, habilitando la entrada de la contraseña para conectarse.
    - Si se selecciona la misma red nuevamente, deshabilita los controles.
8. **Método** `enviar_datos`:
    - Conecta al dispositivo a la red seleccionada utilizando `WifiManager`.
    - Actualiza el estado de conexión en la interfaz y muestra mensajes de éxito o error.
9. **Método** `actualizar_estado_conexion`:
    - Cambia el texto y color del indicador de conexión en la interfaz para reflejar el estado actual (conectado o desconectado).
10. **Método** `refrescar_redes`:
    - Escanea nuevamente las redes disponibles, actualizando la lista y los botones en la interfaz.

7.5. streaming.py

El siguiente código implementa una clase llamada `StreamingInterface`, que proporciona una interfaz gráfica para acceder a plataformas de streaming. Utiliza `tkinter` para la interfaz, `Pillow` para manipular imágenes, y comandos del sistema para abrir un navegador web configurado como aplicación. La clase permite seleccionar una plataforma, abrirla en un navegador, y controlar el cierre del mismo. A continuación, se detallan las funcionalidades generales del código y sus métodos principales.

1. **Clase** `StreamingInterface`:
    - Gestiona la interfaz gráfica para la selección de plataformas de streaming.
    - Permite abrir un navegador web en modo de aplicación para mostrar las plataformas seleccionadas.
    - Controla el cierre del navegador y actualiza la interfaz en consecuencia.
2. **Método** `__init__`:

- Inicializa los atributos de la clase, incluyendo el *callback* para regresar al menú principal.
- Prepara los contenedores y elementos necesarios para manejar la interfaz y el navegador.

3. **Método** `mostrar_interfaz_streaming`:
   - Configura la interfaz gráfica principal con elementos como un fondo, una barra superior y botones para cada plataforma de streaming.
   - Incluye un botón para regresar al menú principal.
4. **Método** `crear_botones_streaming`:
   - Genera dinámicamente botones con íconos representativos de cada plataforma de streaming.
   - Cada botón abre el navegador en la URL correspondiente a la plataforma seleccionada.
5. **Método** `abrir_navegador`:
   - Abre el navegador `chromium-browser` en modo de aplicación para la URL seleccionada.
   - Crea elementos gráficos adicionales como una barra superior para indicar el estado del navegador y un botón para cerrarlo.
6. **Método** `verificar_navegador`:
   - Comprueba si el navegador sigue abierto y actualiza los elementos gráficos en consecuencia.
   - Si el navegador se cierra, elimina los elementos relacionados de la interfaz.
7. **Método** `forzar_cerrar_navegador`:
   - Cierra manualmente el navegador si está abierto.
   - Actualiza la interfaz para eliminar los elementos relacionados con el navegador.
8. **Método** `limpiar_elementos`:
   - Elimina los elementos gráficos relacionados con el navegador, como la barra superior y el botón de cierre.
   - Se utiliza tanto en el cierre manual como automático del navegador.

7.6. media_manager.py

El siguiente código implementa una clase llamada `MediaManager` para gestionar dispositivos de almacenamiento USB y clasificar archivos multimedia en imágenes, audios y videos. Utiliza bibliotecas como `pyudev` para monitorear dispositivos USB, `os` para manipulación de archivos, y `subprocess` para ejecutar comandos del sistema. A continuación, se detalla el propósito general del código y de cada función.

1. **Clase** `MediaManager`:
   - Encapsula la funcionalidad para gestionar medios conectados a través de dispositivos USB.
   - Clasifica los archivos según su formato en imágenes, audios y videos.
2. **Método** `__init__`:
   - Define listas de extensiones de archivo para identificar imágenes, archivos de audio y videos.

- Estas extensiones se utilizan posteriormente para clasificar los archivos en carpetas o dispositivos montados.

3. **Método** `get_media_files`:
   - Escanea un directorio específico y clasifica los archivos en listas separadas de imágenes, audios y videos según las extensiones predefinidas.
   - Retorna tres listas: una para cada tipo de archivo multimedia.
4. **Método** `auto_mount`:
   - Monta automáticamente un dispositivo USB en el sistema utilizando el comando `udisksctl`.
   - Garantiza que el dispositivo esté disponible para acceder a sus archivos.
5. **Método** `get_mount_point`:
   - Utiliza el comando `findmnt` para obtener el punto de montaje de un dispositivo específico.
   - Retorna la ruta donde el dispositivo está montado, si es aplicable.
6. **Método** `monitor_devices`:
   - Monitorea en tiempo real los dispositivos USB conectados al sistema usando `pyudev`.
   - Detecta cuando se conecta un nuevo dispositivo (*add*) y automáticamente lo monta.
   - Clasifica los archivos del dispositivo montado en imágenes, audios y videos, y envía esta información a una cola (`queue`).
   - También detecta la desconexión de dispositivos (*remove*) y actualiza la cola en consecuencia.

7.7. image_slideshow.py

El siguiente código implementa una presentación de imágenes a pantalla completa utilizando `tkinter` para la interfaz gráfica, `Pillow` para la manipulación de imágenes, y `screeninfo` para obtener las dimensiones de la pantalla. La clase `ImageSlideshow` administra tanto la lógica de la presentación como la interfaz gráfica. A continuación, se detallan los aspectos generales del código y sus funciones principales.

1. **Clase** `ImageSlideshow`: Es la clase principal que gestiona la presentación de imágenes.
   - Recibe el *root* de `tkinter`, una lista de rutas de imágenes (`image_paths`) y un *callback* para regresar a la interfaz anterior.
   - Calcula las dimensiones de la pantalla principal utilizando `screeninfo`.
2. **Método** `__init__`:
   - Inicializa los parámetros esenciales, como la lista de imágenes, el índice de la imagen actual, el estado del carrusel (`slideshow_running`) y las dimensiones de la pantalla.
   - Se asegura de preparar los recursos necesarios para la presentación de imágenes.
3. **Método** `mostrar_presentacion`:
   - Configura la interfaz gráfica para mostrar las imágenes en un marco (`Frame`) que ocupa toda la pantalla.

- Incluye un botón que permite regresar a la interfaz anterior llamando al *callback*.
- Llama al método `cambiar_imagen` para iniciar la presentación de imágenes.

4. **Método** `cambiar_imagen`:
   - Cambia la imagen mostrada cada 5 segundos, redimensionándola dinámicamente al tamaño de la pantalla.
   - Actualiza el índice de la imagen actual para permitir una navegación cíclica a través de la lista de imágenes.
   - Usa `after` de `tkinter` para programar el cambio automático.
5. **Método** `volver`:
   - Detiene la presentación de imágenes al cambiar el estado de `slideshow_running` a `False`.
   - Destruye la interfaz actual y llama al *callback* para regresar al menú o interfaz anterior.

7.8. video_usb.py

El siguiente código implementa una clase llamada `VideoUSB`, diseñada para gestionar la selección y reproducción de videos desde un dispositivo USB. La clase proporciona funcionalidades para reproducir videos individuales o en secuencia mediante una presentación automática.

1. **Clase** `VideoUSB`:
   - Gestiona la reproducción de videos utilizando `tkinter` para la interfaz gráfica y `vlc` para la reproducción multimedia.
   - Ofrece opciones para reproducir un video específico o iniciar una presentación automática que recorre todos los videos disponibles.
2. **Método** `__init__`:
   - Inicializa los atributos principales, como la lista de videos, el índice del video actual y los componentes de la interfaz.
   - Llama a `mostrar_interfaz_video` para construir la interfaz gráfica inicial.
3. **Método** `limpiar_frame`:
   - Limpia el contenido del marco actual y detiene cualquier reproducción activa del reproductor.
4. **Método** `mostrar_interfaz_video`:
   - Configura y muestra la interfaz gráfica con botones para cada video disponible y una opción para iniciar la presentación automática si hay múltiples videos.
5. **Método** `crear_botones_videos`:
   - Genera botones dinámicos para cada video en la lista, posicionándolos en filas y columnas.
   - Asigna un comando a cada botón para reproducir el video correspondiente.
6. **Método** `reproducir_video`:
   - Inicia la reproducción del video seleccionado en un marco dedicado dentro de la ventana.

- Configura el reproductor VLC para mostrar el video en una ventana embebida.

7. **Método** `iniciar_presentacion`:
   - Activa la presentación automática, reproduciendo secuencialmente todos los videos en la lista.
8. **Método** `siguiente_video_presentacion`:
   - Gestiona la reproducción del siguiente video en la lista durante una presentación.
   - Finaliza la presentación y regresa a la interfaz principal al completar todos los videos.
9. **Método** `detener_video_y_volver`:
   - Detiene la reproducción del video actual y regresa a la lista de videos.
10. **Método** `volver`:
    - Detiene cualquier reproducción activa y limpia la interfaz antes de regresar al menú principal.

7.9. wifi_manager.py

El siguiente código implementa una clase llamada `WifiManager`, que gestiona la conexión WiFi en un dispositivo utilizando herramientas del sistema como `nmcli` y `socket`. A continuación, se detalla el propósito general del código y de cada función.

1. **Clase** `WifiManager`:
   - Proporciona métodos para verificar si el dispositivo está conectado a Internet, escanear redes WiFi disponibles y conectarse a una red específica.
   - Utiliza herramientas de red estándar como `socket` para la conexión y `nmcli` para la gestión de redes.
2. **Método** `is_connected`:
   - Verifica si el dispositivo tiene acceso a Internet intentando crear una conexión con un servidor DNS público (`8.8.8.8`, de Google).
   - Devuelve `True` si la conexión es exitosa y `False` en caso contrario.
3. **Método** `scan_networks`:
   - Escanea las redes WiFi disponibles utilizando el comando `nmcli`.
   - Extrae y retorna una lista de los nombres de las redes (SSID) detectadas.
   - En caso de error, devuelve una lista vacía.
4. **Método** `connect_to_network`:
   - Intenta conectar el dispositivo a una red WiFi específica utilizando su SSID y contraseña.
   - Utiliza el comando `nmcli` para establecer la conexión.
   - Retorna `True` si la conexión es exitosa y `False` en caso de fallo.

8. Integración de los módulos

Integración de Módulos en el Sistema Multimedia

El sistema multimedia está diseñado como un conjunto modular de clases que trabajan de forma coordinada para proporcionar funcionalidades específicas relacionadas con dispositivos USB, redes WiFi, plataformas de streaming y reproducción multimedia. A continuación, se describe cómo estos módulos se integran y colaboran para ofrecer una experiencia cohesiva al usuario.

- **Coordinación a través de Interfaces**: Las interfaces principales, como `USBInterface`, `RedInterface` y `StreamingInterface`, gestionan la navegación entre las distintas funcionalidades. El uso de métodos como `volver_callback` permite limpiar y actualizar las interfaces dinámicamente, garantizando transiciones fluidas entre las secciones del sistema.

- **Gestión de Dispositivos USB**: La clase `USBInterface` utiliza `MediaManager` para clasificar archivos multimedia en imágenes, música y videos. Posteriormente, delega la gestión de cada tipo de contenido a módulos especializados como `ImageSlideshow`, `AudioUSB` y `VideoUSB`, que proporcionan interfaces gráficas dedicadas para la visualización o reproducción de los archivos.

- **Conectividad y Redes**: La clase `RedInterface` integra el módulo `WifiManager` para escanear redes WiFi, conectarse a ellas y mostrar el estado de la conexión en tiempo real. Esta integración asegura que los cambios en el estado de la red se reflejen dinámicamente en la interfaz gráfica.

- **Streaming en Navegadores**: El módulo `StreamingInterface` permite acceder a plataformas de streaming mediante un navegador en modo de aplicación. Utiliza comandos del sistema operativo para abrir y cerrar el navegador, y actualiza la interfaz gráfica para reflejar el estado del navegador de manera transparente.

9. Conclusiones

La creación de este centro multimedia como proyecto resultó ser un tanto complejo y profundo. Se integraron diversas funcionalidades, como la reproducción de contenido desde un medio extraíble, streaming en línea y la gestión de redes WiFi. Se emplearon herramientas como Tkinter para crear una interfaz intuitiva y fácil de navegar, lo que permitió ofrecer una experiencia de entretenimiento fluida y accesible para los usuarios.

Referencias

[1] G. Galeano, *Programación de sistemas embebidos en C*. Alpha Editorial, 2009.

[2] Raspberry Pi, *Raspberry Pi 4 Model B Datasheet*, Raspberry Pi, 2024, release 1.1, March 2024. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[3] Isaac. (2024, nov) Osmc: el centro multimedia para tu raspberry pi. Hardwarelibre. [Online]. Available: https://www.hwlibre.com/guia-completa-para-apagar-y-reiniciar-raspberry-pi-correctamente/

[4] P. S. Foundation, "Tkinter — python interface to tcl/tk," https://docs.python.org/3/library/tkinter.html, 2024, accessed: 2024-11-26.

[5] P. I. L. Team, "Pillow documentation," https://pillow.readthedocs.io/en/stable/, 2024, accessed: 2024-11-26.

[6] screeninfo Developers, "screeninfo: Get monitor information from the os," https://github.com/rr-/screeninfo, 2024, accessed: 2024-11-26.

[7] pyudev Team, "pyudev — libudev bindings for python," https://pyudev.readthedocs.io/en/latest/, 2024, accessed: 2024-11-26.

[8] T. C. Projects, "Chromium browser," https://www.chromium.org/, 2024, accessed: 2024-11-26.

[9] P. Developers, “Pulseaudio documentation,” https://www.freedesktop.org/wiki/Software/PulseAudio/, 2024, accessed: 2024-11-26.

[10] V. Organization, “Vlc media player,” https://www.videolan.org/doc/, 2024, accessed: 2024-11-26.

[11] G. LLC, “Widevine cdm,” https://www.widevine.com/, 2024, accessed: 2024-11-26.

10. Adicional

Video de prueba: `https://youtu.be/JCH2hj2dtEo?si=ZdE_1C7KUk0pxib1`

Repositorio: `https://github.com/m4rc0m0nt3l0ng0/Proyecto_Final_Centro-Multimedia.git`