

Práctica 5:

Kiosco multimedia

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a configurar la Raspberry Pi como un kiosco multimedia para la reproducción de archivos de audio y video en modo desatendido.

2. Introducción

Cajeros automáticos, mapas interactivos, máquinas de turnos y puntos de venta de autoservicio son algunos ejemplos de kioscos informáticos que ya forman parte de la vida diaria de las personas. Así, los kioscos son una de las aplicaciones más comunes que tienen los sistemas embebidos de gama alta.

Un kiosco integra un conjunto mínimo de periféricos con un procesador de bajo costo para llevar a cabo tareas dedicadas con una interfaz sencilla y amigable para el usuario. Al mismo tiempo, el desempeño del kiosco debe ser óptimo sin subutilizar sus recursos, pero ofreciendo un mínimo de componentes interactivos para evitar el mal manejo del mismo. Es por este motivo que se suelen ocupar tarjetas controladoras de gama media con sistemas operativos que carecen de los elementos clásicos de escritorio que permitirían que cualquier usuario malintencionado pueda hacerse del control del mismo. Para este fin puede utilizarse una Raspberry Pi con un sistema operativo tipo Linux configurado con un gestor gráfico simple como `nodm` (véase la [Sección 3](#)), y un conjunto de paquetes de reproducción de multimedios como `ffmpeg` (véase la [Sección 4](#)) o `vlc` (véase la [Sección 5](#)).

3. NoDM

`nodm` es un manejador gráfico mínimo que inicia sesión gráfica de forma automática con un usuario predeterminado. Al ser un manejador gráfico mínimo `nodm` no integra código para realizar una composición de ventanas sobre un escritorio u operarlas. Es más, no incluye siquiera controles para iniciar sesión o pedir una contraseña.

De acuerdo con el sentido común y con la documentación misma de `nodm`,¹ esto representaría un problema de seguridad en cualquier computadora, pues se da acceso a cualquiera al sistema. No obstante, el autor mismo reconoce que es justamente para esto que se creó `nodm`: para inicializar entornos gráficos en sistemas embebidos tales como dispositivos móviles, paneles de control, y kioscos.

Aunque `nodm` se considera un proyecto abandonado desde 2017 y hoy día manejadores gráficos más pesados como `lightdm` o `gdm` permiten iniciar sesión de forma automática sin necesidad de ingresar contraseña en entornos gráficos de escritorio como *Gnome Desktop* de Ubuntu o XFCE, `nodm` sigue siendo una excelente opción en sistemas embebidos comerciales e industriales por ser extremadamente ligero y rápido, al punto que sigue disponible para su instalación en sistemas basados en Debian como *Raspberry Pi OS Lite* o *Ubuntu Core*.

¹Documentación oficial de `nodm` disponible en <https://manpages.ubuntu.com/manpages/bionic/man8/nodm.8.html>.

4. FFmpeg

De acuerdo con la documentación oficial,² FFmpeg es un proyecto de software gratuito y de código abierto que consiste en un conjunto de bibliotecas y programas para manejar archivos de audio, video y transmisiones multimedia.

El núcleo es la mismísima herramienta de línea de comandos `ffmpeg` diseñada para el procesar y recodificar archivos multimedia. Sin embargo es `ffplay` la herramienta que es de mayor interés en los kioscos multimedia, ya que permite la reproducción de virtualmente cualquier archivo multimedia soportado por FFmpeg, incluyendo virtualmente todas las opciones de transcodificación y transformación aplicables en tiempo real.

La reproducción de archivos es trivial, basta con ejecutar el comando `ffplay` seguido del nombre del archivo a reproducir. Por ejemplo

```
| $ ffplay video.mp4
```

5. VLC media player

De acuerdo con la documentación oficial,³ *VLC media player* es un software de reproductor de archivos multimedia multiplataforma, gratuito y de código abierto, además de ser portátil e incorporar un servidor de transmisión o *streaming* desarrollado por el proyecto VideoLAN. *VLC* es quizá el reproductor multimedia más utilizado por el simple hecho de ser capaz de reproducir virtualmente cualquier archivo multimedia, ya sea audio o video, además de medios físicos como DVDs y BluRays sin necesidad de instalar codecs opcionales.

La reproducción de archivos es trivial, basta con ejecutar el comando `ffplay` seguido del nombre del archivo a reproducir. Por ejemplo

```
| $ vlc video.mp4
```

Una ventaja de *VLC media player* es que puede ser controlado fácilmente desde python con el paquete `python3-vlc`, lo que lo convierte en una herramienta increíblemente poderosa para todo tipo de tareas multimedia.

6. Detección de inserción de dispositivos USB con pyudev

De acuerdo con la documentación oficial,⁴ `pyudev` es un envoltorio hecho en Python puro con licencia LGPL para `libudev`, la biblioteca de información y administración de dispositivos y hardware para Linux. `pyudev` es capaz de enumerar dispositivos, consultar sus propiedades y atributos, e incluso monitorearlos de forma asíncrona.

La detección de nuevos dispositivos es relativamente simple y requiere de únicamente seis pasos: i) importar el paquete `pyudev` en python, ii) crear un objeto de tipo contexto (`Context`) que representa al árbol de dispositivos de la máquina, iii) crear un objeto monitor (`Context`) para la detección de cambios en el contexto dado, iv) configurar al monitor para que inspeccione sólo los dispositivos de interés, v) esperar a que se notifiquen cambios y, finalmente, vi) verificar que el cambio es de tipo inserción y acceder al dispositivo.

Este proceso se ejemplifica con el siguiente código

```
src/miniusbdetect.py
1 import pyudev
2
3 context = pyudev.Context()
4 monitor = pyudev.Monitor.from_netlink(context)
5 monitor.filter_by(subsystem='block', device_type='partition')
6 while True:
7     action, device = monitor.receive_device()
8     if action != 'partition':
9         continue
10    print(f'Detected device {device.sys_name} on {device.sys_path}')
11    break
```

²Documentación oficial de *FFmpeg* disponible en <https://ffmpeg.org/about.html>.

³Documentación oficial de *VLC media player* disponible en <https://www.videolan.org/vlc/>.

⁴Documentación oficial de *pyudev* disponible en <https://pypi.org/project/pyudev/>.

En este ejemplo las líneas 3 y 4 crean los objetos contexto y monitor respectivamente. A continuación la línea 5 configura al monitor para notificar únicamente de cambios en los dispositivos que operan por bloques y que pueden contener particiones, es decir, dispositivos de almacenamiento masivo como discos duros portátiles y memorias USB. A partir de este punto el script está configurado para realizar la detección; por lo que sólo falta realizar la llamada pertinente. Esto se realiza con el método `receive_device()` del objeto `monitor`. Este método bloqueante regresará sólo cuando haya cambios en el árbol de dispositivos y estos coincidan con los filtros establecidos; devolviendo una tupla que contiene el tipo de cambio (inserción o remoción) y un objeto con toda la información del dispositivo.

Como en este ejemplo sólo interesa notificar dispositivos conectados, se ingresa el código de detección en un bucle `while` y se ignoran las acciones que no sean de tipo `add` (agregar o insertar en el árbol de dispositivos). No obstante, en un sistema bien diseñado, ambas acciones `add` y `remove` tendrán códigos de control para llevar a cabo acciones cuando se desconecte un medio, especialmente si se estaba accediendo a la información del dispositivo.

7. Material

Se asume que el alumno cuenta con una Raspberry Pi 3B o posterior con sistema operativo **Raspberry Pi OS Lite** y un monitor conectado a la misma. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Importante

Esta práctica precisa de un sistema mínimo **SIN ESCRITORIO** como *Raspberry Pi OS Lite* o *Ubuntu Core*. Si el sistema instalado es una versión de escritorio o cuenta con un manejador gráfico, no podrá realizar la práctica.

8. Instrucciones

1. Descargue e instale los paquetes necesarios (`nodm`, `ffmpeg` y `vlc`, entre otros).
2. Haga una prueba con `ffplay`
3. Haga una prueba con `vlc`
4. Configure la raspberry para que reproduzca videos desde el arranque en modo desatendido.

8.1. Paso 1: Instalación de paquetes

Antes de instalar paquetes, verifique que su Raspberry esté conectada a internet.

A continuación, actualice el gestor de paquetes `aptitude` con la línea

```
# apt update
```

Posteriormente, proceda a instalar los paquetes necesarios con la línea

```
# apt -y install nodm ffmpeg vlc python3-vlc python3-pyudev wget
```

Durante la instalación de `nodm` se le mostrará una advertencia de seguridad, preguntándole si desea habilitar `nodm` como gestor gráfico. Confirme su decisión.

Finalmente, descargue el video de prueba con la línea:

```
$ wget -qO- https://bit.ly/fsemp05media | tar xvz -C /home/pi
```

8.2. Paso 2: Prueba de reproducción de video con FFmpeg

El siguiente paso consiste en probar la reproducción de video con *ffplay*.

De manera predeterminada la raspberry Pi estará configurada para enviar el canal de audio al monitor HDMI conectado. Como la mayoría de los monitores no cuentan con bocinas integradas, es necesario reconfigurar la raspberry para utilizar la salida de audio estándar con la utilidad de configuración `raspi-config`.

Una vez configurada la salida de audio, reproduzca el video de prueba con la línea

```
$ ffmpeg -i ~/videos/video.mp4
```

En cuestión de segundos verá el video desplegarse en su pantalla. Para terminar la ejecución utilice la combinación de teclas CTRL+C.

8.3. Paso 3: Prueba de reproducción de fotografías con VLC

El siguiente paso consiste en probar la reproducción de fotografías *VLC*. Lo anterior se logra mediante la creación de una lista de reproducción que *VLC* pueda usar para reproducir las fotografías en modo presentación una tras otra.

Para generar dicha lista ejecute los siguientes comandos:

```
$ cd ~/pictures
$ echo "#EXTM3U" > playlist.m3u
$ for f in *.jpg; do realpath "$f" >> playlist.m3u; done
```

Finalmente, inicie la reproducción con *VLC* con la línea:

```
$ vlc playlist.m3u --repeat
```

En cuestión de segundos verá las fotografías desplegarse en pantalla. Para terminar la ejecución utilice la combinación de teclas CTRL+C.

Cabe mencionar que *VLC* permite la reproducción de todos los medios en un directorio simplemente especificando la ruta del directorio en lugar de la ruta de un archivo. ¡Incluso *VLC* es capaz de acceder a contenido multimedia en línea y reproducirlo! Sin embargo, se ha optado por la opción de crear una lista de reproducción a fin de tener más control en el contenido que *VLC* reproduce.

8.4. Paso 4: Configuración de kiosco en modo desatendido

Para configurar la Raspberry Pi en modo kiosco, bastaría con editar el archivo `/etc/rc.local`. Sin embargo, la reproducción de multimedia sin control alguno es prácticamente inútil, por lo que ésta suele delegarse a un script que controle el flujo y permita reaccionar a eventos tales como el toque de la pantalla o de un botón. Aquí es donde entra en juego el *wrapper* de Python para la API de *VLC*.

El uso de la API es increíblemente simple. Una vez instalada, basta con seguir 5 simples pasos: i) importar el paquete `vlc` en python, ii) crear un objeto reproductor o *player*, iii) crear un objeto *media* por cada multimedia a reproducir, iv) cargar el objeto *media* en el reproductor y, finalmente, v) iniciar la reproducción.

src/minikiosk.py

```
1 import vlc
2 player = vlc.MediaPlayer()
3 video = vlc.Media('/home/pi/videos/video.mp4')
4 player.set_media(video)
5 player.play()
```

La función `play()` comienza la reproducción del medio de forma asíncrona y regresa de inmediato, por lo que la ejecución se interrumpirá si el programa termina a continuación. Dependerá del programador controlar durante cuanto tiempo se ejecuta el video, por ejemplo al presionarse un botón. Se puede dejar al video ejecutarse hasta el final consultando la propiedad `is_playing` del objeto `MediaPlayer`.

Finalmente para configurar la ejecución del script cuando la Raspberry Pi reinicie agregue la siguiente línea al archivo `/etc/rc.local` justo antes de la línea de retorno `return 0`.

```
python3 /home/pi/minikiosk.py &
```

9. Experimentos

1. [2 pts] Modifique el archivo de ejemplo `minikiosk.py` para que reproduzca el video de prueba durante 10 segundos e inmediatamente después reproduzca en un bucle infinito cada una de las imágenes de muestra por espacio de 3 segundos.

2. [4 pts] Integre el archivo `minikiosk.py` modificado con el detector de medios extraíbles `usbdetect.py` para que la presentación infinita de imágenes se detenga cuando se inserta una USB y, en lugar de las imágenes de muestra, se reproduzcan las imágenes del medio extraíble.
3. [+2 pts] Modifique el programa anterior para que el video se reproduzca por espacio de 20 segundos. El video debe comenzar con volumen cero que se incrementará gradualmente hasta 100 % durante los primeros 5 segundos, reproducirse a máximo volumen durante 10 segundos y, finalmente, bajar el volumen gradualmente de 100 % a cero durante los últimos 5 segundos.
4. [+5 pts] Modifique el programa anterior para que la lista de reproducción sea controlable utilizando los pines GPIO2, GPIO3, GPIO4, GPIO17, GPIO27 y GPIO22 como anterior, siguiente, parar, pausar/resumir volumen+ y volumen- respectivamente.

10. Cuestionario

1. [1.0pt] ¿Qué funciones o métodos se utilizan para subir y bajar el volumen durante una reproducción? Ilustre con código de ejemplo.
2. [1.0pt] ¿Qué funciones o métodos se utilizan para ejecutar la reproducción en modo de pantalla completa? Ilustre con código de ejemplo.
3. [2.0pt] ¿Cuáles son las funciones que permiten desplazarse en una lista de reproducción y controlarla (siguiente, anterior, pausa, detener y reanudar)? Ilustre con código de ejemplo

A. Programa Ejemplo: minikiosk.py

minikiosk.py

```
1 import vlc
2 import time
3
4 player = vlc.MediaPlayer()
5 video = vlc.Media('/home/pi/videos/video.mp4')
6 player.set_media(video)
7 player.play()
8 while player.is_playing:
9     time.sleep(0)
```

B. Programa Ejemplo: usbdetect.py

usbdetect.py

```
1 import os
2 import pyudev
3 from time import sleep
4 import subprocess as sp
5
6 def print_dev_stats(path):
7     photos = []
8     for file in os.listdir(path):
9         if file.endswith(".jpg") \
10            or file.endswith(".png"):
11             photos.append(file)
12     print("{} has {} photos.".format(path, len(photos)))
13 #end def
14
15 def print_dev_info(device):
16     print("Device sys_path: {}".format(device.sys_path))
17     print("Device sys_name: {}".format(device.sys_name))
18     print("Device sys_number: {}".format(device.sys_number))
19     print("Device subsystem: {}".format(device.subsystem))
20     print("Device device_type: {}".format(device.device_type))
21     print("Device is_initialized: {}".format(device.is_initialized))
22 #end def
23
24 def auto_mount(path):
25     args = ["udisksctl", "mount", "-b", path]
26     sp.run(args)
27 #end def
28
29 def get_mount_point(path):
30     args = ["findmnt", "-unl", "-S", path]
31     cp = sp.run(args, capture_output=True, text=True)
32     out = cp.stdout.split(" ")[0]
33     return out
34 #end def
35
36 context = pyudev.Context()
37 monitor = pyudev.Monitor.from_netlink(context)
38 monitor.filter_by(subsystem="block", device_type="partition")
39 while True:
40     action, device = monitor.receive_device()
41     if action != "add":
42         continue
43     print_dev_info(device)
44     auto_mount("/dev/" + device.sys_name)
45     mp = get_mount_point("/dev/" + device.sys_name)
46     print("Mount point: {}".format(mp))
47     print_dev_stats(mp)
48     break
```
