

Práctica 4:

Control remoto de la Raspberry Pi via WiFi y servidor web

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

Martes 25 de Febrero, 2020

1. Objetivo

El alumno aprenderá a configurar la Raspberry Pi como punto de acceso inalámbrico que permita acceder a un servidor web simple que controle el puerto GPIO de la misma.

2. Material

Se asume que el alumno cuenta con una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Si se cuenta con una Raspberry Pi sin WiFi integrado (e.j Raspberry Pi2), se precisará de un adaptador WiFi USB compatible para la misma.

Además, el alumno necesitará el alambrado de la [Práctica 3](#).

3. Instrucciones

1. Alambre el circuito tal y como se detalla en la [Práctica 3](#).
2. Realice los ejercicios y experimentos de la [Práctica 3](#).

3.1. Paso 1: Configuración de la Raspberry Pi como punto de acceso inalámbrico

Para operar como punto de acceso la Raspberry Pi necesita tener instalado el software apropiado, incluyendo un servidor DHCP para proporcionar a los dispositivos que se conecten una dirección IP.

Se comienza por instalar los paquetes DNSMasq y HostAPD:

```
# apt-get install dnsmasq hostapd
```

Si están ejecutándose los servicios, deténgalos a fin de poder reconfigurarlos

```
# systemctl stop dnsmasq
# systemctl stop hostapd
```

3.1.1. Configuración del adaptador y el cliente DHCP

Para configurar una red independiente con servidor DHCP la Raspberry Pi debe tener asignada una dirección IP estática en el adaptador inalámbrico que proveerá la conexión. Debido a que la Raspberry Pi tiene un procesador pequeño, se configurará para servir en una red privada clase C, es decir con direcciones IP del tipo 198.168.x.x. Así mismo, se supondrá que el dispositivo inalámbrico utilizado es wlan0.

Para configurar la dirección IP estática edite el archivo de configuración `/etc/dhcpd.conf` como superusuario:

```
interface wlan0
    static ip_address=192.168.1.254/24
    nohook wpa_supplicant
```

A continuación, reinicie el cliente DHCP

```
# service dhcpcd restart
```

3.1.2. Configuración del servidor DHCP

. El siguiente paso consiste en configurar el servidor DHCP, provisto por el servicio dnsmasq.

De manera predeterminada el archivo de configuración `/etc/dnsmasq.conf` contiene mucha información que no es necesaria, por lo que es más fácil comenzar desde cero. Respáldelo y cree uno nuevo con el siguiente texto:

```
1 # Use the require wireless interface - usually wlan0
2 interface=wlan0
3 # Reserve 20 IP addresses, set the subnet mask, and lease time
4 dhcp-range=192.168.1.200,192.168.1.220,255.255.255.0,24h
```

Esta configuración proporcionará 20 direcciones IP entre 192.168.1.200 y 192.168.1.220, válidas durante 24 horas. Ahora debe iniciarse el servidor DHCP

```
systemctl start dnsmasq
```

3.1.3. Configuración del punto de acceso

. Para configurar el punto de acceso se debe editar el archivo de configuración `/etc/hostapd/hostapd.conf` con los parámetros adecuados.

Respáldelo y cree uno nuevo con el siguiente texto:

```
1 # Wireless interface
2 interface=wlan0
3 # Specification: IEEE802.11
4 driver=nl80211
5 # The SSID or name of the network
6 ssid=Raspbberri
7 # Password of the network
8 wpa_passphrase=12345678
9 wpa=2
10 wpa_key_mgmt=WPA-PSK
11 wpa_pairwise=TKIP
12 # Mode and frequency of operation
13 hw_mode=g
14 # Broadcast channel
15 channel=5
16 wmm_enabled=0
17 macaddr_acl=0
18 auth_algs=1
19 ignore_broadcast_ssid=0
20 rsn_pairwise=CCMP
```

La configuración ingresada configura la Raspberry Pi para crear una red inalámbrica tipo 802.11g en el canal 5 de nombre *Raspbberri* y contraseña 12345678 con seguridad WPA2.

Los modos de operación posibles son:

- a = IEEE 802.11a (5 GHz)
- b = IEEE 802.11b (2.4 GHz)
- g = IEEE 802.11g (2.4 GHz)

Importante: Tanto el nombre de la red o SSID y la contraseña no deben entrecomillarse. La contraseña debe tener entre 8 y 64 caracteres. **Cambie el SSID a Raspberry_Apellido para evitar conflictos.**

Ahora edite el archivo `/etc/default/hostapd` y reemplace la línea que comienza con `#DAEMON_CONF` con:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

3.1.4. Configuración e inicio del punto de acceso

. Finalmente, habilite los servicios para iniciar el punto de acceso:

```
# systemctl unmask hostapd
# systemctl enable hostapd
# systemctl start hostapd
```

Verifique que los servicios se están ejecutando

```
# systemctl status hostapd
# systemctl status dnsmasq
```

3.2. Paso 2: Configuración de la Raspberry Pi como servidor Web

Raspbian es una variante d Debian, por lo que se le dará bien servir páginas web de forma segura, especialmente cuando se utiliza Apache. Sin embargo, configurar Apache para enlazarse con Python y operar la GPIO no es una tarea trivial, por lo que en esta práctica se utilizará un servidor web simple basado en el `BaseHTTPRequestHandler` que incorpora el paquete `http.server` de Python.

Para habilitar un servidor web en Python, basta con heredar de la clase `BaseHTTPRequestHandler` e implementar el método `do_GET` para que imprima el código HTML al socket vía el método `self.wfile.write` tal como se muestra en el [Código de Ejemplo 1](#).

```
1 from http.server import BaseHTTPRequestHandler, HTTPServer
2
3 class WebServer(BaseHTTPRequestHandler):
4     def do_GET(self):
5         self.send_response(200)
6         self.send_header("Content-type", "text/html")
7         self.end_headers()
8         self.wfile.write(bytes("<html><body>Hola Mundo!!!</body></html>", "utf-8"))
9
10 def main():
11     webServer = HTTPServer(("192.168.1.254", 80), WebServer)
12     print("Servidor iniciado")
13     print("\tAtendiendo solicitudes entrantes")
14     try:
15         webServer.serve_forever()
16     except KeyboardInterrupt:
17         pass
18     webServer.server_close()
19     print("Server stopped.")
```

Código ejemplo 1: Archivo `simple-webserver.py`

Script de Python presentado inicia un servidor web que atiende todas las peticiones entrantes vía la interfaz con la IP 192.168.1.254 (el punto de acceso) en el puerto 80 (HTTP predeterminado). A cada petición se le devolverá una señal de estado HTTP200 u *OK*, seguido por código HTML. Es importante aclarar que para cada archivo servido se debe especificar el tipo de archivo en la cabecera.

Genere el archivo `simple-webserver.py` y ejecútelo. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: <http://192.168.1.254>.

Con ligeras modificaciones es posible servir cualquier tipo de archivo. Todas las peticiones ingresadas en la barra de direcciones del navegador llegarán por método *GET*, por lo que deberán ser procesadas en el método `do_GET`, accediendo al atributo de clase `self.path`, relativo al directorio de trabajo. En caso de que no se proporcione un archivo, `do_GET` tendrá que proporcionar la página por defecto, típicamente nombrada `index.html`, pero que en este caso por motivos didácticos se ha nombrado `user_interface.html` (véase [Código de Ejemplo 2](#)).

```
1 def do_GET(self):
2     # Revisamos si se accede a la raiz.
3     # En ese caso se responde con la interfaz por defecto
4     if self.path == '/':
5         # 200 es el código de respuesta satisfactorio (OK)
6         # de una solicitud
```

```

7     self.send_response(200)
8     # La cabecera HTTP siempre debe contener el tipo de datos mime
9     # del contenido con el que responde el servidor
10    self.send_header("Content-type", "text/html")
11    # Fin de cabecera
12    self.end_headers()
13    # Por simplicidad, se devuelve como respuesta el contenido del
14    # archivo html con el código de la página de interfaz de usuario
15    self._serve_ui_file()
16    # En caso contrario, se verifica que el archivo exista y se sirve
17    else:
18        self._serve_file(self.path[1:])

```

Código ejemplo 2: Método do_GET del archivo webserver.py

Para servir un archivo se tiene que verificar que el éste exista, proporcionar su tipo mime en la cabecera y devolver los datos como una cadena binaria. Esto se realiza en el método interno `_serve_file`. Si el archivo no se encontrare, se devuelve un error *HTTP404* como se muestra en el [Código de Ejemplo 3](#):

```

1    def _serve_file(self, rel_path):
2        if not os.path.isfile(rel_path):
3            self.send_error(404)
4            return
5        self.send_response(200)
6        mime = magic.Magic(mime=True)
7        self.send_header("Content-type", mime.from_file(rel_path))
8        self.end_headers()
9        with open(rel_path, 'rb') as file:
10            self.wfile.write(file.read())

```

Código ejemplo 3: Método `_serve_file` del archivo webserver.py

La interacción cliente servidor se lleva a cabo de manera similar. Dependerá de si los datos se envían por método *GET* o *POST*, de los cuales se prefiere el segundo pues hace más difícil inyectar datos. De manera análoga se utiliza el método `do_POST` que recibe y procesa los datos. En esta práctica, se utilizan datos codificados mediante JSON para hacer llamadas asíncronas del cliente y sin respuesta por parte del servidor (véase [Código de Ejemplo 4](#)).

```

1    def do_POST(self):
2        # Primero se obtiene la longitud de la cadena de datos recibida
3        content_length = int(self.headers.get('Content-Length'))
4        if content_length < 1:
5            return
6        # Después se lee toda la cadena de datos
7        post_data = self.rfile.read(content_length)
8        # Finalmente, se decodifica el objeto JSON y se procesan los datos.
9        # Se descartan cadenas de datos mal formados
10       try:
11           jobj = json.loads(post_data.decode("utf-8"))
12           self._parse_post(jobj)
13       except:
14           print(sys.exc_info())
15           print("Datos POST no reconocidos")

```

Código ejemplo 4: Método `do_POST` del archivo webserver.py

El método `do_POST` preentado en el [Código de Ejemplo 4](#) interpreta los datos recibidos como cadenas de texto unicode de 8 bits (*utf-8*) que contienen objetos en JSON que son decodificados a un diccionario de Python. El diccionario es después enviado al método interno `_parse_post` mostrado en el [Código de Ejemplo 5](#) que analiza los datos y realiza las acciones pertinentes.

```

1    def _parse_post(self, json_obj):
2        if not 'action' in json_obj or not 'value' in json_obj:
3            return
4        switcher = {
5            'led' : leds,
6            'marquee' : marquee,
7            'numpad' : bcd

```

```

8     }
9     func = switcher.get(json_obj['action'], None)
10    if func:
11        print('\tCall{}({})'.format(func, json_obj['value']))
12        func(json_obj['value'])

```

Código ejemplo 5: Método `_parse_post` del archivo `webserver.py`

Genere los archivos `webserver.py` y `user_interface.html` (véase [Apéndices A y B](#)), luego ejecute el script de Python. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: `http://192.168.1.254`. Debería ver una pantalla similar a la siguiente.

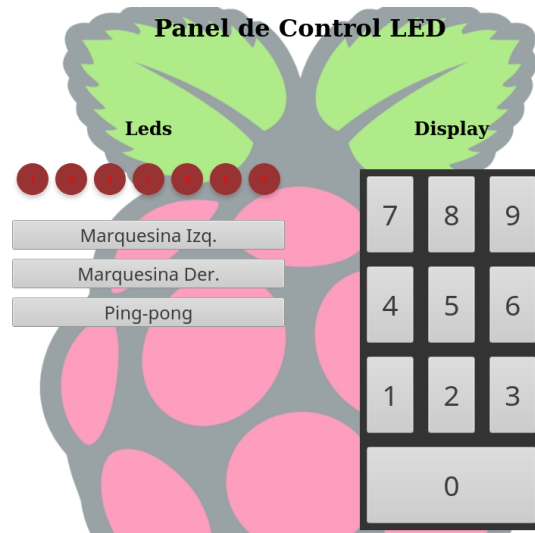


Figura 1: Caption: Intefaz de usuario del controlador de Leds en la Raspberry Pi.

4. Experimentos

Integre el código de la [Práctica 3](#) en un archivo python llamado `led_manager.py` y que ofrezca las siguientes funciones:

1. [2 pts] Encendido del del 1–7 al presionar el boton adecuado
2. [2 pts] Desplegado de la marquesina izquierda al presionar el boton adecuado
3. [2 pts] Desplegado de la marquesina derecha al presionar el boton adecuado
4. [2 pts] Desplegado de la marquesina tipo ping-pong al presionar el boton adecuado
5. [2 pts] Desplegado del dígito correcto en el display de 7 segmentos al presionar el boton correspondiente

A. El archivo `webserver.py`

```
1 import os
2 import sys
3 import json
4 import magic
5 from led_manager import leds, bcd, marquee
6 from http.server import BaseHTTPRequestHandler,
  HTTPServer
7 # import time
8 # import time
9
10 # Nombre o dirección IP del sistema anfitrión del
  servidor web
11 # address = "localhost"
12 address = "192.168.1.254"
13 # Puerto en el cual el servidor estará atendiendo
  solicitudes HTTP
14 # El default de un servidor web en producción debe ser 80
15 port = 8080
16
17
18 class WebServer(BaseHTTPRequestHandler):
19     """Sirve cualquier archivo encontrado en el servidor
       """
20     def _serve_file(self, rel_path):
21         if not os.path.isfile(rel_path):
22             self.send_error(404)
23             return
24         self.send_response(200)
25         mime = magic.Magic(mime=True)
26         self.send_header("Content-type", mime.from_file(
            rel_path))
27         self.end_headers()
28         with open(rel_path, 'rb') as file:
29             self.wfile.write(file.read())
30
31     """Sirve el archivo de interfaz de usuario"""
32     def _serve_ui_file(self):
33         if not os.path.isfile("user_interface.html"):
34             err = "user_interface.html not found."
35             self.wfile.write(bytes(err, "utf-8"))
36             print(err)
37             return
38         try:
39             with open("user_interface.html", "r") as f:
40                 content = "\n".join(f.readlines())
41         except:
42             content = "Error reading user_interface.html"
43             self.wfile.write(bytes(content, "utf-8"))
44
45     def _parse_post(self, json_obj):
46         if not 'action' in json_obj or not 'value' in
           json_obj:
47             return
48         switcher = {
49             'led': leds,
50             'marquee': marquee,
51             'numpad': bcd
52         }
53         func = switcher.get(json_obj['action'], None)
54         if func:
55             print('\tCall{ }({})'.format(func, json_obj['value']
           ))
56             func(json_obj['value'])
57
58
59     """do_GET controla todas las solicitudes recibidas vía
       GET, es
60     decir, páginas. Por seguridad, no se analizan
       variables que lleguen
61     por esta vía"""
62     def do_GET(self):
63         # Revisamos si se accede a la raíz.
64         # En ese caso se responde con la interfaz por
           defecto
65         if self.path == '/':
66
67             # 200 es el código de respuesta satisfactorio (OK)
68             # de una solicitud
69             self.send_response(200)
70             # La cabecera HTTP siempre debe contener el tipo
           de datos mime
71             # del contenido con el que responde el servidor
72             self.send_header("Content-type", "text/html")
73             # Fin de cabecera
74             self.end_headers()
75             # Por simplicidad, se devuelve como respuesta el
           contenido del
76             # archivo html con el código de la página de
           interfaz de usuario
77             self._serve_ui_file()
78             # En caso contrario, se verifica que el archivo
           exista y se sirve
79         else:
80             self._serve_file(self.path[1:])
81
82
83     """do_POST controla todas las solicitudes recibidas vía
       POST, es
84     decir, envíos de formulario. Aquí se gestionan los
       comandos para
85     la Raspberry Pi"""
86     def do_POST(self):
87         # Primero se obtiene la longitud de la cadena de
           datos recibida
88         content_length = int(self.headers.get('Content-
           length'))
89         if content_length < 1:
90             return
91         # Después se lee toda la cadena de datos
92         post_data = self.rfile.read(content_length)
93         # Finalmente, se decodifica el objeto JSON y se
           procesan los datos.
94         # Se descartan cadenas de datos mal formados
95         try:
96             jobj = json.loads(post_data.decode("utf-8"))
97             self._parse_post(jobj)
98         except:
99             print(sys.exc_info())
100             print("Datos POST no reconocidos")
101
102     def main():
103         # Inicializa una nueva instancia de HTTPServer con el
104         # BaseHTTPRequestHandler definido en este archivo
105         webServer = HTTPServer((address, port), WebServer)
106         print("Servidor iniciado")
107         print("\tAtendiendo solicitudes en http://{ }:{ }".
           format(
108             address, port))
109
110         try:
111             # Mantiene al servidor web ejecutándose en segundo
           plano
112             webServer.serve_forever()
113         except KeyboardInterrupt:
114             # Maneja la interrupción de cierre CTRL+C
115             pass
116         except:
117             print(sys.exc_info())
118         # Detiene el servidor web cerrando todas las
           conexiones
119         webServer.server_close()
120         # Reporta parada del servidor web en consola
121         print("Server stopped.")
122
123
124     # Punto de anclaje de la función main
125     if __name__ == "__main__":
126         main()
```

Código ejemplo 6: Archivo `webserver.py`

B. El archivo `user_interface.html`

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Panel de Control LED - Raspberry Pi</title>
5 <meta charset="ISO-8859-1">
6 <style type="text/css">
7   html{
8     width: 100vw;
9     height: 100vh;
10    min-width: 100vw;
11    min-height: 100vh;
12    margin: 0;
13    padding: 0;
14    box-sizing: border-box;
15    overflow: hidden;
16  }
17
18  body{
19    width: 800px;
20    height: 100vh;
21    max-width: 800px;
22    min-height: 100vh;
23    padding: 0;
24    margin: 0 auto;
25    box-sizing: border-box;
26  }
27
28  body::after{
29    content: "";
30    position: absolute;
31    top: 0;
32    left: 0;
33    bottom: 0;
34    right: 0;
35    z-index: -1;
36    opacity: 0.5;
37    background-image: url('img/raspberry.png');
38    background-repeat: no-repeat;
39    background-attachment: fixed;
40    background-position: center;
41    background-size: contain;
42  }
43
44  header{
45    width: 100%;
46    padding: 0;
47    margin: 0;
48    box-sizing: border-box;
49    text-align: center;
50  }
51
52  h1{
53    height: 2em;
54    padding: 1em 0;
55  }
56
57  .container{
58    width: 100%;
59    padding: 0;
60    margin: 0;
61    box-sizing: border-box;
62    display: flex;
63    flex-direction: row;
64  }
65
66  .column{
67    flex: 1 0 0;
68    display: flex;
69    flex-direction: column;
70  }
71
72  .numpad{
73    width: 6.5em;
74    height: 12.75em;
75    background-color: #333333;
76    margin: 0.5em auto;
77    padding: 0;
78    font-size: 28pt;
79    flex-wrap: wrap;
80  }
81
82  .numbutton{
83    font-size: inherit;
84    flex: 1 0 0;
85    margin: 0.25em;
86  }
87
88  .ledstrip{
89    justify-content: space-evenly;
90    width: 90%;
91    height: 4em;
92    padding: 0;
93    margin: 0.5em auto;
94    /*background-color: #333333;*/
95  }
96
97  .ledbutton{
98    width: 3.5em;
99    height: 3.5em;
100   color: #F00;
101   background-color: #933;
102   border-radius: 50%;
103   margin: 0.25em;
104   padding: 0;
105   border: none;
106   box-shadow: 0px 4px 5px rgba(0, 0, 0, 0.2);
107 }
108
109 .ledbutton:hover{
110   background-color: #F33;
111 }
112
113 .widebutton{
114   font-size: 18pt;
115   width: 90%;
116   margin: 0.25em auto;
117 }
118
119 .on{
120   color: #0F0;
121   background-color: #3F3;
122 }
123
124 </style>
125 </head>
126 <body>
127   <header><h1>Panel de Control LED</h1></header>
128   <section class="container">
129     <article class="column">
130       <header><h2>Leds</h2></header>
131       <section class="container ledstrip">
132         <button class="ledbutton" onclick="handle(this, 'led', 1)">1</button>
133         <button class="ledbutton" onclick="handle(this, 'led', 2)">2</button>
134         <button class="ledbutton" onclick="handle(this, 'led', 3)">3</button>
135         <button class="ledbutton" onclick="handle(this, 'led', 4)">4</button>
136         <button class="ledbutton" onclick="handle(this, 'led', 5)">5</button>
137         <button class="ledbutton" onclick="handle(this, 'led', 6)">6</button>
138         <button class="ledbutton" onclick="handle(this, 'led', 7)">7</button>
139       </section>
140       <button class="widebutton" onclick="handle(this, 'marquee', 'left')">Marquesina Izq.</button>
141       <button class="widebutton" onclick="handle(this, 'marquee', 'right')">Marquesina Der.</button>
142       <button class="widebutton" onclick="handle(this, 'marquee', 'pingpong')">Ping-pong</button>
143     </article>
```

```

144 <article class="column">
145   <header><h2>Display</h2></header>
146   <section class="container numpad">
147     <button class="numbutton" onclick="handle(this,
148       'numpad', 7)">7</button>
149     <button class="numbutton" onclick="handle(this,
150       'numpad', 8)">8</button>
151     <button class="numbutton" onclick="handle(this,
152       'numpad', 9)">9</button>
153     <button class="numbutton" onclick="handle(this,
154       'numpad', 4)">4</button>
155     <button class="numbutton" onclick="handle(this,
156       'numpad', 5)">5</button>
157     <button class="numbutton" onclick="handle(this,
158       'numpad', 6)">6</button>
159     <button class="numbutton" onclick="handle(this,
160       'numpad', 1)">1</button>
161     <button class="numbutton" onclick="handle(this,
162       'numpad', 2)">2</button>
163     <button class="numbutton" onclick="handle(this,
164       'numpad', 3)">3</button>
165     <button class="numbutton" onclick="handle(this,
166       'numpad', 0)">0</button>
167   </section>
168 </article>
169 </section>
170 </body>
171 </html>
172 <script language="javascript">
173 <!--
174 function deactivateAll(){
175   var buttons = document.getElementsByTagName('button');

```

```

176   for(button in buttons)
177     button.classList.remove("on")
178 }
179
180 function activate(sender){
181   if(sender == null)
182     return;
183   sender.classList.add("on");
184 }
185
186 function handle(sender, action, value){
187   // deactivateAll();
188   // activate(sender);
189   submit(action, value);
190 }
191
192 function submit(action, value){
193   var xhr = new XMLHttpRequest();
194   xhr.open("POST", window.location.href, true);
195   xhr.setRequestHeader('Content-Type', 'application/json');
196   xhr.send(JSON.stringify({
197     'action' : action,
198     'value' : value,
199   }));
200 }
201 //-->
202 </script>

```

Código ejemplo 7: Archivo user_interface.html