Proyecto de uso público

1. INDICE

2. INTRODUCCIÓN.

El presente proyecto pretende ayudar a los técnicos que instalan la tarjeta Raspberry Pi, sin acceso físico o en instalación remota, con la incorporación de un sistema de encendido/apagado que sea útil y eficaz para mantener el equipo encendido ante perdidas de alimentación y la posibilidad de encender/apagar remotamente.

3. CREACIÓN E INSTALACIÓN DEL SOFTWARE.

La Raspberry Pi incorpora una funcionalidad oculta para poder encenderla sin necesidad de desconectar físicamente el cable USB y evitar fallos futuros, en el conector, que consiste en puentear, con un pulsador NO, la patilla 5 (GPIO3) del conector de expansión con la patilla 6 (GND). El presente proyecto no puede evitar el uso de esta patilla para encender la Raspberry Pi ya que esta implementado, de fabrica, en el mismo SOC

Deseamos implementar un sistema de apagado y encendido seguro para la Raspeberry Pi en caso de perdida, total o parcial, de la alimentación.

Buscando información se ha encontrado, que el pin 5, correspondiente al puerto GPIO 3 (SCL I2C), esta preparado para ese cometido y añadiendo un script hecho con python las siguientes lineas, podemos controlar el apagado mediante pulsador.

El problema de este método es que el bus I2C queda anulado ya que la patilla 5 queda configurada como entrada. Para poder evitar todos los inconvenientes generados con este método, se propone el diseño de un sistema que vigile la alimentación y un script que controle otra patilla distinta para apagar pero manteniendo el encendido por patilla 5 (GPIO3), algo que ya viene instalado de fabrica. Podemos escribir y ejecutar el script:

```
import RPi.GPIO as GPIO
from subprocess import call

LED = 17     #Patilla 11
PULSADOR = 27      #Patilla 13
GPIO.setmode(GPIO.BCM)

GPIO.setup(PULSADOR, GPIO.IN)
GPIO.wait_for_edge(PULSADOR, GPIO.FALLING)
#para activar el led del pulsador, si lo lleva
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED, GPIO.HIGH)
time.sleep(3)       #un pequeño sleep


GPIO.output(LED, GPIO.LOW)
call(['shutdown', '-h', 'now'], shell=False)
```

La primera prueba que realizamos consiste en probar el encendido de la Raspberry Pi después de un apagado estándar tipo >init 0 o Shutdown. Una vez detenidos todos los procesos y con el led ámbar apagado, el rojo encendido ya que no se debe de desconectar la tarjeta de la alimentación pulsamos el botón de encendido. La Raspberry Pi enciende correctamente y a continuación, ejecutamos el programa python y pulsamos de nuevo para apagarla. El programa funciona correctamente si apaga la Raspberry Pi.

Para utilizar el mismo botón para ambos eventos hemos de unir, mediante diodos, el pulsador a las patillas GPIO3 y GPIO27

Una vez verificado el programa, abrimos con el nano un fichero txt donde escribiremos las lineas del script que ejecutará el programa, en segundo plano, al encender el equipo.
El script escrito para la ocasión es el siguiente:

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:     ScriptAutoApagado
# Required-Start:   $all
# Required-Stop:    $syslog
# Default-Start:    2 3 4 5
# Default-Stop:     0 1 6
# Short-Description:   Apagar la Raspberry automaticamente
# Description:      Script de inicio para apàgar la Raspberry
#           tras producirse un evento de corte de alimentación
### END INIT INFO
sudo python3 /home/pi/Scripts/AutoApagado.py
```

El proceso completo para transformar el script en ejecutable, una vez guardado, se describe a continuación. Desde la terminal enviamos el script a la carpeta /etc/init.d escribiendo

$ sudo cp <nombre del Script sin extensión> /etc/init.d

nos trasladamos a la carpeta /etc/init.d

$ cd /etc/init.d

verificamos que el programa esté en la carpeta y esté escrito en negro, si lo deseamos, con:

$ ls

ahora vamos a darle los permisos correspondientes para convertirlo en ejecutable:

$ sudo chmod +x <nombre del Script sin extensión>

podemos cargar el proceso, sin reiniciar, mediante:

$ sudo update-rc.d <nombre del Script sin extensión> defaults

Si tuviéramos problemas, en este punto, podemos ejecutar la linea:

$ sudo update-rc.d <nombre del Script sin extensión> remove

y volvemos a ejecutar la línea con “defaults”. Verificamos la carga del script con el comando:

$ sudo chkconfig -list

veremos todos los procesos cargados y su situación on/off por capas de ejecución.
Podemos activar cualquier proceso con el comando:

o desactivar con:
$ sudo chkconfig <nombre del Script sin extensión> off

También podemos borrar un scripts mediante el comando

$ sudo update-rc.d <nombre del Script sin extensión> remove

o borrar el archivo de la carpeta

$ sudo rm -r <nombre del Script sin extensión>

La única forma de verificar su funcionamiento es, ejecutar manualmente el programa python para apagar el equipo, encenderlo con el pulsador y una vez encendido, volver a apagarlo mediante el pulsador, sin ejecutar el programa python directamente

4. ENVIO DE EMAIL

Para poder advertir a los técnicos y usuarios que el sistema ha fallado, existe la posibilidad de enviar un email para realizar acciones correctivas.

Se crea un archivo con los destinatarios deseados y se instala la libreria smtplib, si no está instalada, mediante

$ pip install smtplib

o, desde el shell de python, podemos ejecutar

>>> help('smtplib')

Para averiguar si está instalada.
Añadiremos a nuestro programa el código necesario. Primero la función de apertura y lectura del fichero de destinatarios donde escribiremos el nombre, coma y el correo, en cada fila uno por ejemplo:

Antonio Martinez,amart@rasp.com
Susana Sainz,ssainz@rasp.com
técnico,tecnico@subcontrata.es
todos falsos

```
def cargar_destinatarios(archivo_destinatarios):
    lista = []#
    archivo = open(textfile, 'r')    #abrimos el archivo
    lineas = archivo.readlines()    #cargamos el contenido en la variable lineas
    archivo.close()
    for line in lineas:
        destinatario = {} #los separamos por tabulaciones
        destinatario = line.split(",")    #esto nos corta las lineas separando en 2 lo que se encuentra a cada lado de la coma. A continuacion limpiamos y elimínanos los caracteres que no deseamos. Suponemos que la lista está escrita a mano y no de forma automatica, pero el \n split no lo elimina.
        lista.append(destinatario) #añadimos a nuestra lista
    try:
        if len(lista)>1:
            for x in lista:
                x[1] = x[1].replace("\n","")
                x[1] = x[1].replace("\t","")
    except:
        pass
        return destinatarios
```

Para garantizar la seguridad, añadiremos otro archivo con los datos que necesitamos para el envío de los e-mail con la siguiente estructura:

Nombre de quien envia
login

Proyecto de uso público

password
numero de serie del equipo que envia
Asunto del email

Extraeremos los datos de este fichero y los preparamos para su uso con la función de carga de usuario

```
def carga_usuario(archivo_usuario):
    usuario = []
    archivo = open(archivo_usuario, 'r')
    lineas = archivo.readlines()
    archivo.close()
    for x in lineas:
        dato = x.replace("\n","")
        dato = dato.replace("\t","")
        usuario.append(dato)
    return usuario
```

A continuación una función que envía el email

```
def send_email(servidor, puerta, login, password, msg)
                        server = smtplib.SMTP(servidor, puerta)
                        server.startls()
                        server.login(usuario, palabradepaso)
                        server.sendmail(msg['From'],msg['to'], msg.as_string())
                        server.quit()
```

Con esto enviaremos un mensaje a un destinatario por lo que necesitaremos otra función que nos vaya cargando cada destinatario. Esta será la función principal o "main", si el programa no se ejecuta como módulo o "enviar" para usarlo como módulo

```
#def main(Adestinatario='
def enviar(Adestinatario='
    global user
    global email
    global password
    global asunto
    global equipo
    listado = cargar_destinatarios(
    datos_usuario = carga_usuario(Ausuario)
    #print(len(datos_usuario))
    user=datos_usuario[0]
    login=datos_usuario[1]
    password=datos_usuario[2]
    equipo = datos_usuario[3]
    asunto = datos_usuario[4]
    print('Usuario ',  user)
    print('email ',  login)
    print('pass ' )

    # Recorre los destinatario
```

```
    for destinatario in listado:
        if len(destinatario)==2:
            mensaje = ' mensaje automatico para '+ destinatario[0] + ' !! \n'\
            'Fallo de seguridad en el sistema. Comuniquese con el responsable de area \n'\
            'Equipo=: '+ equipo
            msg = MIMEText(mensaje)
            msg['Subject'] = asunto
            msg['From'] = user
            msg['To'] = destinatario[1]
            send_email('smtp.gmail.com', 587, login, password, msg)

            print ('mensaje :'+ mensaje)
            print ('Enviado a ' + destinatario[1] + ' por '+user)
    return True
#if __name__ == "__main__":
#    main()
```

4.ALTERNATIVA AL INIT.D

Existe UNA ALTERNATIVA AL Init.d más segura para el uso de scripts que es el uso del systemd. Para ello, nos trasladamos a la carpeta mediante:

$sudo su

$cd /usr/lib/systemd/system

$nano <nombre>.service

escribimos:

[Unit]
Description=Programa para el apagado del equipo
After=network.target

[Service]
Type=simple
User=root
ExecStart=/usr/bin/python3 /ruta del programa <otro nombre o el mismo>.py
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=<nombre>
KillMode=process

[Install]
WantedBy=multi-user.target
Alias=pulsador.service
Crtl-O ->Intro
Crtl-x

ahora toca cargarlo en el sistema para ser ejecutado:

systemctl enable <nombre>.service

podemos conocer el staTus del daemon escribiendo:

systemctl status <nombre>.service

o recargarlo MEDIANTE

systemctl restart <nombre>.service