multitâche avec Raspberry Pi

Savoir-faire : commande de feux de circulation

Dogan Ibrahim (Royaume Uni)

Voici un exemple de projet extrait d'ouvrage publié récemment par Elektor. Intitulé *Multitasking with RPi*, ce livre est destiné aux étudiants, aux ingénieurs en exercice et aux amateurs intéressés par le les projets multitâches utilisant le langage Python3 sur l'ordinateur Raspberry Pi (RPI).

Le multitâche est devenu l'un des sujets les plus importants dans les systèmes à base de µC, notamment dans les automatismes. Avec l'accroissement de leur complexité, on attend plus des projets, ce qui, sur le même système, impose l'utilisation de plusieurs tâches interdépendantes et partageant l'unité centrale (ou de multiples UC) pour effectuer les opérations. D'où l'importance croissante du fonctionnement multitâche dans les applications à base de µC. Beaucoup de projets d'automatisme complexes font appel à un noyau multitâche. Dans le livre, on utilise le langage Python avec le Raspberry Pi (RPI). On peut aussi utiliser d'autres modèles de RPi sans rien changer au code.

Le livre repose sur des projets. Son objectif principal est d'enseigner les principes de base du multitâche avec Python sur RPi. Il présente de nombreux projets dûment testés avec les modules multitâches de Python. Chaque projet est décrit et commenté en détail. Les listages complets des programmes sont fournis pour chaque projet. Les lecteurs devraient pouvoir les utiliser tels quels, ou les adapter à leurs propres besoins.

Savoir-faire : commande de feux de circulation

Ce projet consiste à concevoir une commande simple des feux de circulation d'un carrefour, à l'intersection de deux routes : *East Street* et *North Street*. Il y a des feux à chaque branche du carrefour. Des boutons d'appel pour piétons se trouvent près des feux sur *North Street*. L'appui sur un bouton d'appel provoque le passage au rouge de tous les feux à la fin de leurs cycles. Un signal sonore est alors déclenché pour indiquer aux piétons qu'ils peuvent traverser sans danger. Un afficheur LCD indique le cycle *piétons* ou *circulation* en cours. La **figure 9.12** montre la disposition au carrefour.

Pour ce projet, les durées fixes suivantes sont affectées à chaque feu, ainsi que la durée du signal sonore pour piétons. Pour simplifier, on suppose que les deux routes du carrefour ont les mêmes durées (en secondes) :

- rouge : 19 s
- orange : 2 s
- vert : 15 s
- orange + rouge : 2 s
- piétons : 10 s

La durée totale du cycle des feux est fixée à 38 s.

La séquence adoptée pour ces feux de circulation est la suivante (qui n'est peut-être pas celle dont vous êtes familier dans votre pays) : Rouge ➞ Orange+Rouge ➞ Vert - Orange... Vert ➞ Orange ➞ Rouge ➞ Orange+Rouge...

La **figure 9.13** montre le diagramme du projet, la **figure 9.14** le schéma. Les LED rouge (R), orange (O) et verte (V) sont utilisées pour représenter les vrais feux de circulation.

Les connexions suivantes sont réalisées entre le RPi et les éléments de signalisation routière :

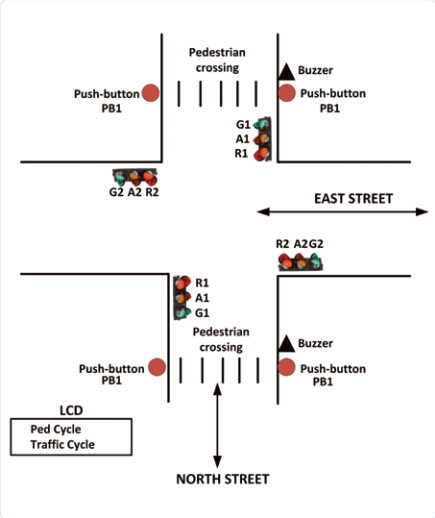

Figure 9.12. Signalisation au carrefour.

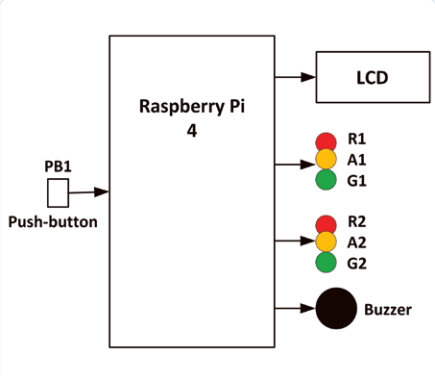

Figure 9.13. Diagramme du projet.

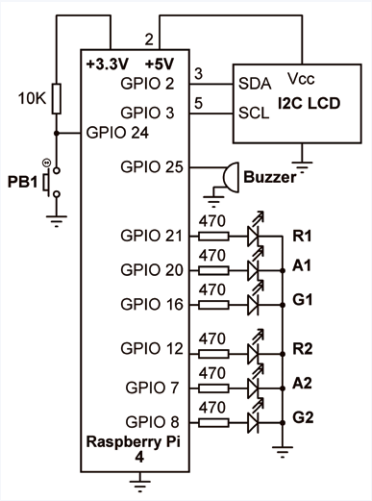

Figure 9.14. Schéma du circuit.

Raspberry Pi	Signalisation
GPIO 21	LED R1
GPIO 20	LED A1
GPIO 16	LED G1
GPIO 12	LED R2
GPIO 7	LED A2
GPIO 8	LED G2
GPIO 25	signal sonore
GPIO 2	LCD SDA
GPIO 3	LCD SCL
GPIO 24	PB1 (bouton d'appel)

La **figure 9.15** montre le listage complet (nom du programme : traffic.py ; téléchargement [1]). En début de programme, on importe les modules nommés *RPi, time, I²C LCD driver* et *multiprocessing* et on crée deux files nommées `pedq` et `lcdq`.
Deux fonctions nommées `ONOF` et `CONF_OUT` sont définies dans le programme. La fonction `ONOF` a deux arguments : `port` et `state`. Cette fonction envoie l'état (0 ou 1) au port GPIO spécifié. La fonction `CONF_OUT` a un paramètre nommé *port*. Cette fonction configure le port GPIO spécifié en mode sortie.
Il y a deux processus dans le programme : `Lights` (= feux) et `Pedestrian`.(= piéton) Au début du processus `Lights`, on définit les connexions entre le RPi, les LED (feux de circulation) et le buzzer puis ces ports sont configurés en sorties. De plus, tous ces ports sont mis à zéro pour éteindre toutes les LED et le buzzer. Enfin, les LED sont séquencées

Figure 9.15: Listage du programme traffic.py

```
#-------------------------------------------------------------------
#                    Commande de feux de circulation
#                    ===============================
#
# Il s'agit d'un projet de commande des feux de circulation d'un
# carrefour. On utilise 6 LED pour représenter les feux de
   circulation.
# On utilise de plus un bouton pour le passage des piétons, et un
# afficheur LCD indique en permanence l'état des feux.
#
# Auteur  : Dogan Ibrahim
# Fichier : traffic.py
# Date    : Mai 2020
#-------------------------------------------------------------------
import RPi.GPIO as GPIO                      # Importe RPi
import multiprocessing                       # Importe multiprocessing
import time                                  # Importe time
import RPi_I2C_driver                        # Bibliothèque I2C
LCD = RPi_I2C_driver.lcd()                   # Importe LCD
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                       # GPIO en mode BCM
pedq = multiprocessing.Queue()               # Création de la file pedq
lcdq = multiprocessing.Queue()               # Création de la file lcdq

#
# Cette fonction envoie la donnée " state " (0 ou 1) au port spécifié
#
def ONOF(port, state):
   GPIO.output(port, state)

#
# Cette fonction configure le port spécifié en sortie
#
def CONF_OUT(port):
   GPIO.setup(port, GPIO.OUT)

#
# Processus de commande des feux
#
def Lights():                                # Processus Lights
   R1=21; A1=20; G1=16                       # Connexions des LED
   R2=12; A2=7;  G2=8                        #
   Buzzer=25                                 # Connexion du buzzer
   CONF_OUT(R1); CONF_OUT(A1); CONF_OUT(G1)  # Configuration en mode
   CONF_OUT(R2); CONF_OUT(A2); CONF_OUT(G2)  # sortie et mise à zéro
   CONF_OUT(Buzzer)                          # de tous les ports
   ONOF(R1,0); ONOF(A1,0); ONOF(G1,0); ONOF(R2,0); ONOF(A2,0); ONOF(G2,0)
   ONOF(Buzzer, 0)

   RedDuration = 15
   GreenDuration = 15
   AmberDuration = 2

#
# Commande de la séquence des feux de circulation
#
   while True:                               # Boucle infinie
      ONOF(R1,0); ONOF(A1,0); ONOF(G1,1); ONOF(R2,1); ONOF(A2,0);
    ONOF(G2,0)
      time.sleep(RedDuration)
      ONOF(G1,0); ONOF(A1,1)
      time.sleep(AmberDuration)
      ONOF(A1,0); ONOF(R1,1); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(R2,0); ONOF(G2,1)
      time.sleep(GreenDuration)
```

```
      ONOF(G2,0); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(A1,1); ONOF(R2,1)
      time.sleep(AmberDuration)

      while not pedq.empty():                # Si appel piétons
         lcdq.put(1)                         # Mise en file lcdq
         ONOF(G1,0); ONOF(R1,1); ONOF(A1,0)  # Feu 1 rouge seul
         ONOF(G2,0); ONOF(R2,1); ONOF(A2,0)  # Feu 2 rouge seul
         d = pedq.get()                      # Vide la file pedq
         ONOF(Buzzer, 1)                     # Active le buzzer
         time.sleep(10)                      # Attend 10 secs
         ONOF(Buzzer, 0)                     # Désactive le buzzer
         d = lcdq.get()                      # Vide la file lcdq


def Pedestrian():                            # Processus Pedestrian
   PB1 = 24
   GPIO.setup(PB1, GPIO.IN)                  # PPB1 en mode entrée

   while True:                               # Boucle infinie
      while GPIO.input(PB1) == 1:            # Attente PB1 appuyé
         pass
      pedq.put(1)                            # Mise en file pedq
      while GPIO.input(PB1) == 0:            # Attente PB1 relâché
         pass

#
# Création des processus
#
p = multiprocessing.Process(target = Lights, args = ())
q = multiprocessing.Process(target = Pedestrian, args = ())
p.start()
q.start()

#
# Commande de LCD. Affichage de " Ped Cycle " ou de  " Traffic Cycle "
#
LCD.lcd_clear()                              # Effacement du LCD
LCD.lcd_display_string("TRAFFIC CONTROL", 1) # En-tête

while True:                                  # Boucle infinie
   if not lcdq.empty():
      LCD.lcd_display_string("Ped Cycle    ", 2)
   else:
      LCD.lcd_display_string("Traffic Cycle", 2)
   time.sleep(1)
```

Figure 9.16. Exemple d'affichage.

dans le bon ordre avec les bonnes durées. Vers la fin de la fonction, on vérifie si le bouton *piétons* a été pressé. Si c'est le cas, la file `pedq` n'est pas vide. Pendant le cycle *piétons*, les feux rouges des deux rues sont allumés pour arrêter la circulation et céder le passage aux piétons.
Le buzzer est également activé pendant 10 s pendant le cycle *piétons* pour les avertir qu'ils peuvent traverser sans danger.
Le processus `Pedestrian` surveille en permanence le bouton PB1. Si le bouton est pressé, un « 1 » est envoyé à la file `pedq` pour que le processus *Lights* puisse facilement détecter cette action et commencer le cycle *piétons*.
Le programme principal commande l'afficheur LCD. Au démarrage, le message « TRAFFIC CONTROL » est affiché sur la première ligne.
La seconde vérifie en permanence la file `lcdq` et affiche soit « Ped Cycle » (cycle *piétons*) soit « Traffic Cycle » (cycle *circulation*).
La **figure 9.16** montre un exemple d'affichage sur l'afficheur LCD.

200381-03

LIEN

[1] **téléchargement du programme traffic.py** : www.elektormagazine.fr/200381-03