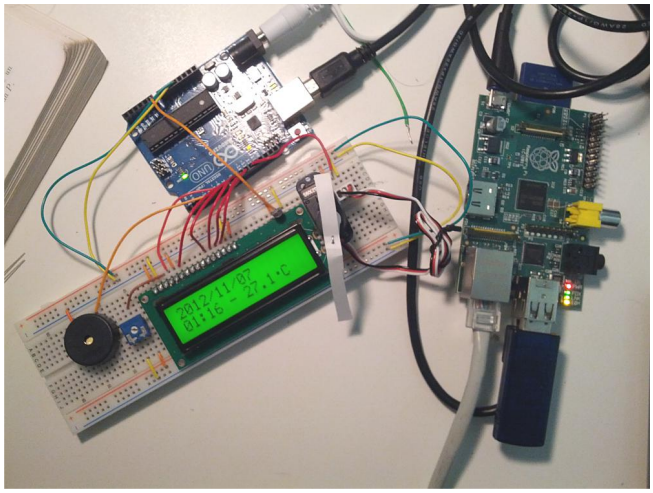
Le monde extérieur

Je vais maintenant vous montrer comment faire communiquer l'Arduino avec le monde extérieur grâce au Raspberry Pi. Pour que vous compreniez, nous allons fabriquer une horloge moderne, capable de mesurer la température extérieure, avec une alarme initialisée via bluetooth (avec un appareil Android dans ce cas) et une actualisation de la date et de l'heure via un serveur ntp...



Le projet avec ses instructions, une application Android et les composants nécessaires se trouvent ici : **https://github.com/nanpy/eggsamples/tree/master/synclock**. Pour illustrer le fonctionnement de Nanpy dans un contexte multithreadé, ce programme crée un fil d'exécution (thread) pour chaque fonctionnalité, et les affiche tous sur le même ACL. Dans cet article, je ne montre que la partie à l'intérieur des boucles "while True" de chaque méthode "run", par conséquent, je vous recommande de le suivre avec le code source. Commençons par le fil principal, TimeThread, qui lit l'heure depuis notre serveur ntp chaque seconde puis la stocke dans une variable globale, milltime :

```
response = ntplib.NTPClient().request(
      'europe.pool.ntp.org',
      version=3)
milltime = int(response.tx_time)
```

Pour afficher la date et l'heure sur l'ACL, créez un second fil d'exécution, ShowTimeThread :

```
...
self.servo = Servo(12)
...
dt = datetime.fromtimestamp(milltime)
lcd.printString(dt.strftime('%Y/%m/%d'),
      0, 0)
lcd.printString(dt.strftime('%H:%M'),
      0, 1)
self.servo.write(90 + (30 * self.c))
self.c *= -1
```

Chaque seconde, nous récupérons la variable globale milltime, la transformons dans un format lisible puis affichons la date et l'heure sur l'ACL. Comme vous pouvez le constater, printString peut être appelée en spécifiant la position (colonne, ligne) où vous souhaitez voir apparaître la chaîne sur l'ACL. Ensuite, nous actionnons le servo-moteur comme une horloge chaque seconde. La température peut être mise à jour dans un autre fil. Lisons la valeur de notre capteur de température depuis la broche analogique 0 et affichons-la sur l'ACL près de l'heure, toutes les 60 secondes :

```
temp = ((Arduino.analogRead(0) / 1024.0)
      * 5.0 - 0.5) * 100
lcd.printString("- %0.1f\xDFC" % temp,
      6, 1)
```

Voyons à présent comment communiquer avec un téléphone Android pour définir l'alarme avec le bluetooth. Je l'ai associé au Raspberry Pi avant de démarrer grâce au guide : **http://wiki.debian.org/BluetoothUser**. Rappelez-vous d'installer python-bluez aussi. Nous utiliserons AlarmClock, une classe thread-safe, pour enregistrer et récupérer la valeur de l'alarme sur disque (voir code). Nous pouvons alors lancer notre communication bluetooth dans un fil AlarmClockThread :

```
...Connexion et init Bluetooth...
cli_sock, cli_info = srv_sock.accept()
cli_sock.send("%d:%d:%d", ck.getAlarm())
try:
    while True:
        data = cli_sock.recv(3)
        if len(data) == 0: break
        ck.setAlarm(ord(data[0]),
                    ord(data[1]),
                    ord(data[2]))
except IOError:
    pass
```

Notre Raspberry Pi se comporte en serveur, attendant une connexion bluetooth : après que cela soit arrivé, il envoie l'heure d'alarme à notre appareil et attend une nouvelle valeur à enregistrer. Dans TimeThread, nous comparons l'heure actuelle et la valeur de l'alarme : si elles sont égales, nous pouvons démarrer un autre fil, PlayAlarmThread, qui joue la note Do pendant 250 ms, à 5 reprises, en utilisant un objet à travers un haut-parleur contrôlé par la 4e broche numérique. Debout !

Commencez à imaginer vos propres projets avec Nanpy, par exemple en ressuscitant votre vieille voiture radiocommandée : youtu.be/NI4PDfVMdgM

Article de Andrea Stagi