C Programming on Raspberry Pi

Extrait : communiquer par Wi-Fi

Dogan Ibrahim (Royaume-Uni)

Le Raspberry Pi est traditionnellement programmé en Python. Bien que Python soit un langage très puissant, le langage C est probablement le plus utilisé par les programmeurs, et pas seulement parce qu'il est enseigné dans pratiquement toutes les écoles et universités techniques. Cet article donne un avant-goût du C sur Raspberry Pi, afin de vous aider à évaluer l'effort que vous devez fournir pour développer vos propres projets matériels. Vous pouvez travailler depuis votre laboratoire personnel, avec un livre et votre Raspberry Pi préféré à portée de main comme support de cours. Voyons ce qui se passe en C !

***Note de l'éditeur** : cet article est un extrait du livre de 376 pages, intitulé « C Programming on Raspberry Pi – Develop innovative hardware-based projects in C » formaté et légèrement modifié pour correspondre aux normes éditoriales et à la mise en page du magazine Elektor. Puisque cet article est extrait d'une publication plus vaste, certains termes peuvent faire référence à des passages du livre d'origine situés ailleurs. L'auteur et l'éditeur ont fait de leur mieux pour l'éviter et seront heureux de répondre aux questions – pour les contacter, voir l'encadré « Des questions, des commentaires ? ».*

UDP et TCP/IP

La communication par liaison Wi-Fi se déroule entre un client et un serveur. On utilise des *sockets* pour envoyer et recevoir des paquets de données. Le dispositif côté serveur attend généralement une connexion des clients et une fois celle-ci établie, la communication bidirectionnelle peut commencer. On utilise principalement deux protocoles pour envoyer et recevoir des paquets de données sur une liaison Wi-Fi : UDP et TCP. TCP est un protocole qui nécessite l'établissement d'une connexion, ce qui garantit la livraison des paquets. Les paquets reçoivent des numéros de séquence et la réception de tous les paquets fait l'objet d'un accusé de réception. La numérotation sert à remettre les paquets dans l'ordre s'ils ont suivi des chemins différents. À cause du système d'acquittement, TCP est généralement lent mais fiable, car il garantit la livraison des paquets. UDP, quant à lui, est un protocole sans connexion préalable. Les paquets n'ont pas de numéro de séquence et, par conséquent, il n'y a aucune certitude qu'ils arriveront à destination. UDP est moins surchargé que TCP et il est donc plus rapide. Le **tableau 1** énumère quelques différences entre les protocoles TCP et UDP.

Tableau 1. Communications par paquets TCP et UDP.

TCP	UDP
Les paquets ont des numéros de séquence et la livraison de chaque paquet fait l'objet d'un accusé de réception.	Il n'y a pas d'accusé de réception
Lent	Rapide
Pas de perte de paquets	Des paquets peuvent être perdus
Surcharge importante	Surcharge faible
Nécessite plus de ressources	Nécessite moins de ressources
Établissement d'une connexion	Sans connexion
Plus difficile à programmer	Plus facile à programmer
Exemples : HTTP, HTTPS, FTP	Exemples : DNS, DHCP, jeux vidéo

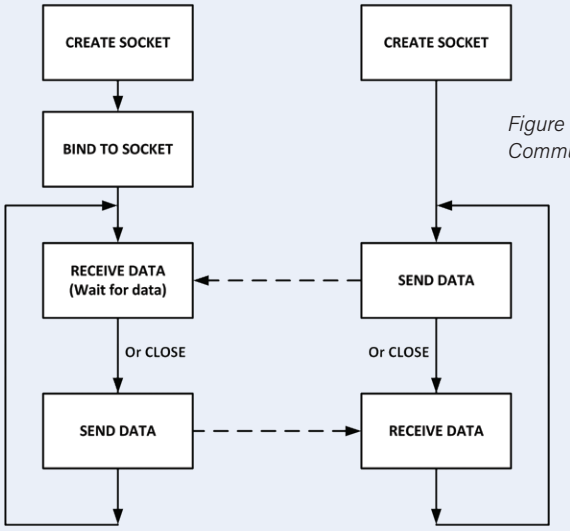

Figure 1. Communication UDP.

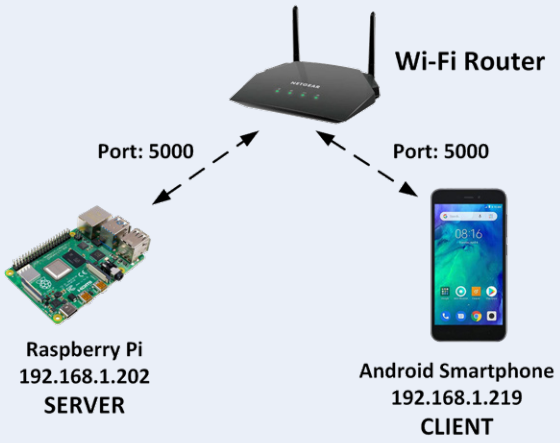

Figure 2. Schéma fonctionnel du projet.

Communication UDP

La **figure 1** montre la communication UDP sur une liaison Wi-Fi.

Serveur

1. Créer un *socket* UDP.
2. Lier le *socket* à l'adresse du serveur.
3. Attendre que le paquet de données arrive du client.
4. Traiter le paquet de données.
5. Envoyer une réponse au client, ou fermer le *socket*.
6. Retourner à l'étape 3 (si le *socket* n'est pas fermé).

Client

1. Créer un *socket* UDP.
2. Envoyer un message au serveur.
3. Attendre la réponse du serveur.
4. Répondre au processus.
5. Retourner à l'étape 2, ou fermer le *socket*.

Projet 1 : communiquer avec un ordiphone Android en utilisant UDP (le RPi est le serveur)

Dans ce projet, on utilise le protocole UDP pour recevoir et envoyer des données vers/depuis un ordiphone Android. Ici, le RPi est le serveur et l'ordiphone Android le client.

Objectif : ce projet vise à montrer comment une liaison UDP peut être établie entre un RPi et un ordiphone Android.

Schéma fonctionnel : la **figure 2** présente la structure générale du projet. Le RPi et l'ordiphone Android communiquent par l'intermédiaire d'une liaison routeur Wi-Fi.

Listage des programmes : le **listage 1** présente le programme *MyServer.c* qui peut être extrait du paquetage logiciel du livre. Rendez-vous sur le site (*www.elektor.fr/19703*), faites défiler la page vers le bas jusqu'à *Téléchargements*, puis cliquez sur *Software_C Programming on Raspberry Pi*. Stockez le fichier .zip localement, puis extrayez tous les fichiers ou seulement *MyServer.c*.

Les fichiers d'en-tête nécessaires au programme sont inclus au début de *MyServer.c*. Le message *Hello from Raspberry Pi* est stocké dans le tableau de caractères `msg`. Un *socket* de type UDP est ensuite créé en appelant la fonction `socket` et en enregistrant son identifiant

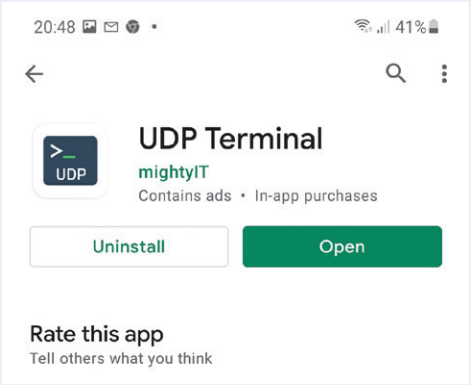

Figure 3. Appli de terminal UDP sur un ordiphone Android.

Figure 4. Entrez les numéros de port.

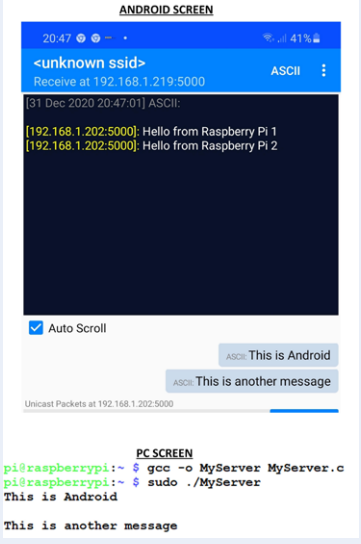

Figure 5. Exemple d'exécution du programme.

dans la variable `sock`. Les informations sur l'ordinateur serveur (RPi) sont ensuite données, l'adresse étant définie comme `INADDR_ANY`, de sorte que tout autre ordinateur sur le même réseau avec le numéro de port réglé sur 5000 pourra établir la communication avec le RPi. Les informations de l'ordinateur serveur sont ensuite liées au port spécifié en appelant la fonction `bind`.

Le reste du programme s'exécute dans une boucle. Dans celle-ci, on appelle la fonction `recvfrom` pour attendre la réception d'un paquet de données de l'ordinateur client (ordiphone Android). Remarquez qu'il s'agit d'un appel bloquant et que la fonction attendra jusqu'à la réception de données du client. Le programme se termine si le caractère `x` est reçu de l'ordinateur client. Un caractère `NULL` est ajouté aux données reçues qui sont affichées sur l'écran du RPi sous forme d'une chaîne de caractères à l'aide de la fonction `printf`. Une variable entière appelée `count` est convertie en caractère et ajoutée à la fin du tableau de caractères `msg`. Celui-ci est envoyé à l'ordinateur client. Dans la première la boucle, l'ordinateur client affichera *Bonjour de Raspberry Pi 1*. La deuxième fois, il affichera *Bonjour de Raspberry Pi 2*, et ainsi de suite. Le *socket* est fermé juste avant la fin du programme. Le programme peut être compilé et exécuté comme suit

```
gcc -o MyServer MyServer.c
sudo ./MyServer
```

Il existe de nombreuses applications UDP disponibles gratuitement sur le Play Store. Dans ce projet, on utilise le Terminal UDP de mightyIT (voir **fig. 3**). Saisissez le numéro de port et l'adresse IP de votre ordiphone Android et cliquez sur *Start Terminal*, comme indiqué à la **figure 4**. Un exemple d'exécution du programme est présenté à la **figure 5**, où l'on voit à la fois l'écran de l'ordiphone et celui du PC.
Remarque : vous pouvez trouver l'adresse IP de votre RPi en utilisant la commande `ifconfig`.

210346-04

Listage 1. MyServer.c

```
/*-------------------------------------------------------------------
   COMMUNICATION RASPBERRY PI - ORDIPHONE ANDROID
   ==============================================
Il s'agit d'un programme UDP. Le programme reçoit puis envoie des
messages à un ordiphone via le socket UDP. Le programme se termine
lorsque le caractère x est envoyé depuis l'ordiphone. Ceci est le
programme serveur UDP, qui communique sur le port 5000.
Auteur : Dogan Ibrahim
Fichier : MyServer.c
Date : Décembre 2020
-------------------------------------------------------------------*/
   #include <netinet/in.h>
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <unistd.h>
   #define Port 5000
   #define BUFFSIZE 1024
//
    // Début du programme MAIN
    //
   int main(void)
   {
     int sock, len, num, count = 1 ;
     char buffer[BUFFSIZE] ;
     struct sockaddr_in serveraddr, clientaddr ;
     char msg[26] = "Bonjour de Raspberry Pi" ;
  sock = socket(AF_INET, SOCK_DGRAM, 0) ;
     len = sizeof(clientaddr) ;
  serveraddr.sin_family = AF_INET ;
     serveraddr.sin_addr.s_addr = INADDR_ANY ;
     serveraddr.sin_port = htons(Port) ;
     bind(sock,(struct sockaddr *)&serveraddr, sizeof(serveraddr))
   ;
  while(1)
     {
       num = recvfrom(sock, buffer, BUFFSIZE,MSG_WAITALL,
             struct sockaddr *)&clientaddr, &len) ;
       if(buffer[0] == 'x')break ;
       buffer[num] = '\0' ;
       printf("%s\n",buffer) ;
           sprintf(&msg[24], "%d",count) ;
       count++ ;
       sendto(sock, &msg, strlen(msg), MSG_CONFIRM,
       struct sockaddr *)&clientaddr, len) ;
     }
     close(sock) ;
   }
```

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (d.ibrahim@btinternet.com) ou contactez Elektor (redaction@elektor.fr).

Contributeurs

Texte : Dogan Ibrahim
Rédaction : Jan Buiting
Mise en page : Harmen Heida
Traduction : Denis Lafourcade