Guide du TP

Cyrille Pierre, Alexis Pereda

12 septembre 2019

Le sujet du TP explique comment fonctionne le module de messagerie que vous devez réaliser. Il contient toutes les informations techniques nécessaires pour que le système puisse fonctionner. Cependant, on pourrait penser que beaucoup d'informations sont manquantes. Cela vient du fait qu'il y a beaucoup de détails dont l'implémentation est *libre*. Vous pouvez donc réaliser votre système comme vous le souhaitez. Les seuls limitations sont :

— l'écriture du code en C ou en shell
— l'utilisation d'outils propre à Linux (IPC, threads, daemon, ...)

Il a donc une grande liberté sur la façon de réaliser le module de messagerie. Et si on ne sait pas trop comment s'y prendre ou quels outils utiliser, cela peut sembler compliqué à mettre en place. Ce document est donc là pour vous donner des idées sur la procédure de développement de votre système embarqué. Pour commencer, le document présente un exemple d'utilisation du système pour mieux vous rendre compte de son fonctionnement. Dans un deuxième temps il découpe le travail à faire en tâches qui pourront être partagées par les membres du binôme. Pour finir, un exemple complet d'implémentation est présenté pour ceux qui ne veulent pas trop réfléchir à une architecture à mettre en place.

1 Exemple d'utilisation

Voici un exemple de scénario possible pour l'utilisation du module de messagerie. Cet exemple n'a pas pour but d'être exhaustif mais seulement d'illustrer un fonctionnement classique du système. Pour bien comprendre cet exemple, il est nécessaire d'avoir lu au moins la première partie du sujet.

Imaginons que le serveur de communication est en place et que 2 autres Raspberry sont déjà connectées à ce serveur. Elles se sont indentifiées sous les noms de *cheval* et *mouton*. Maintenant nous allumons notre Raspberry. Une fois qu'elle a fini de démarrer, les programmes de gestion de la messagerie se lancent automatiquement. La Raspberry va donc se connecter au serveur et s'identifier sous le nom de *chevre*. Le port série est fonctionnel donc le PC va pouvoir lui envoyer des instructions pour envoyer ou consulter des messages.

Nous souhaitons d'abord consulter les messages déjà lues. Pour cela, il faut faire une requête depuis le PC vers la Raspberry (en passant par le port série). La Raspberry nous renvoie 4 messages qui viennent d'une utilisation précédente du système (car les messages ont été conservés après l'extinction de la Raspberry).

Maintenant nous écrivons un nouveau message que l'on va envoyer à *mouton*. Encore une fois, cela se fait par l'intermédiaire du PC. Le message est d'abord transmis à la Raspberry. Après analyse de la requête, la Raspberry envoie ensuite ce message au serveur en respectant le protocole de ce dernier. Le serveur reçoit le message et détecte que le destinataire est *mouton*. Il envoie donc le message à la Raspberry *mouton* qui fera alors clignoter sa led verte pour indiquer qu'il y a un message non lu.

Un peu plus tard, *cheval* décide d'envoyer un message à tous le monde (grâce au destinataire particulier : *all*). Le serveur va reçevoir la requête de *cheval* et va ensuite envoyer le contenu du message à toutes les Raspberry actuellement connectées : *cheval*, *mouton* et *chevre*. Les leds vertes des 3 Raspberry vont donc clignoter (celle de mouton clignotait déjà). On se rend compte que notre Raspberry (*chevre*) à reçu un message et nous décidons de le consulter. La requête va encore une fois se faire par l'intermédiaire du PC et la Raspberry affichera l'ensemble des messages non lus. Dans le cas précis, il s'agira uniquement du message de *cheval*. La Raspberry va également mettre à jour l'état de ce dernier message pour le passer de l'état *non lu* à l'état *lu*.

2 Organisation du travail à faire

Le fonctionnement du système de messagerie n'est pas très compliqué à comprendre mais il nécessite cependant un peu de travail à faire. Ce système doit être capable de communiquer en réseau, en port série, de manipuler des leds, de stocker des données, ...Pour que la réalisation du système soit facilement apréhendable, il est nécessaire de découper le travail en tâches. C'est pourquoi je vous propose un exemple de découpage du projet. Il faut cependant garder à l'esprit que ce que je vous présente ici est ma vision personnelle du projet. Je ne vous empêche pas de découper le travail de façon complètement différente.

Les tâches présentés ici ne donne pas d'informations sur la façon dont elles sont implémentées. Il peut s'agir d'un même programme pour plusieurs tâches ou à l'inverse, plusieurs programmes pour une même tâche.

2.1 La gestion des Leds

Dans ce projet, l'interface de la Raspberry est assez limitée. Les seuls informations visuelles qu'elle transmet directement passe par des leds. Celles-ci peuvent être allumées, éteintes ou clignotante (pour la verte). Une partie du travail de ces TPs sera donc de maîtriser le fonctionnement des Leds et de proposer une interface d'utilisation. Voici quelques contraintes utiles :

- **La configuration de `sysfs` est automatique** L'interface sysfs est complexe. Il faut donc encapsuler ce mécanisme pour le limiter uniquement à l'utilisation des GPIOs sous forme de leds.
- **L'interface est simple à utiliser.** L'idéal serait de pouvoir appeler une simple fonction C ou une commande shell pour changer l'état d'une led.
- **Pas d'attente lorsqu'on change l'état d'une led.** Le script ou la fonction rend tout de suite la main après avoir changer l'état de la led. C'est simple à faire pour les états allumé ou éteint mais c'est plus compliqué pour le cas du clignotement.

2.2 Communication avec le serveur

Le serveur utilise une socket TCP pour établir une connexion avec les Raspberry. Pour dialoguer avec lui, il faut utiliser le protocole qui a été définit dans le sujet du TP. Étant donné que le serveur existe déjà et est partagé par tous les binômes, il est impossible d'adapter le protocole à son utilisation. Pour cette partie du projet, il sera beaucoup plus simple d'utiliser le C. Pour la création de la socket TCP cliente, vous avez un code disponible sur `https://zz.cpierre.ovh/linux_emb/tp/`.

Voici quelques contraintes utiles :

- **Lorsqu'un message est reçu du serveur, il est tout de suite traité.** Cela signifie qu'il faut écouter en permanence le serveur pour reçevoir les nouveaux messages.
- **Les messages reçus sont stockés avec l'état *non-lu*.** Il est important de pouvoir savoir que le message n'a pas encore été lue. Cette information doit donc être disponible.
- **Certaines requêtes du PC nécessite l'envoi d'un message du serveur.** Il n'y a pas que le serveur TCP à écouter. Il faut aussi gérer les requêtes venant du client. Ces requêtes devront être transformées pour valider le protocole du serveur avant d'être envoyées.
- **Après l'envoi d'une requête, le code d'erreur doit être traité** Lorsque l'on envoie une commande au serveur, il faut vérifier qu'elle s'est exécutée correctement. Pour cela, on utilise le code d'erreur qui est renvoyé par le serveur. Il faut donc lire dans la socket juste après avoir

écrit. Attention, il n'est cependant pas possible d'avoir plusieurs threads/processus qui lisent en même temps la socket.

2.3 Gestion des données

La Raspberry va manipuler des messages. Ces messages possèdent au minimum un destinataire, un expéditeur et un contenu. Mais on peut également ajouter une date, un état (lu, non-lu, à envoyer, ...). Toutes ces informations ont besoin d'être stockées quelque part. Cela peut être des fichiers textes, XML, YAML, csv, ..., mais aussi une base de donnée ou encore une structure de donnée (en C). Ces données seront consultées ou créées par un ou plusieurs processus.

Voici quelques contraintes utiles :

— **Les données doivent être persistantes.** Si l'on redémarre les programmes ou la Raspberry, les messages ne sont pas perdu. Il ne faut donc pas les stocker exclusivement dans la RAM.
— **L'accès aux données doit être *thread safe*.** Il faut contrôler l'accès en lecture / écriture des messages par les processus pour éviter les conflits.
— **L'organisation des données doit être efficace.** Avant de commencer à mettre en place la gestion des données, il faut bien réfléchir à la structure utilisée. Avec une bonne structure, la manipulation des messages sera très simple. Une interface en C ou en shell pour manipuler les messages peut être une bonne idée pour cacher la complexité de la structure.

2.4 Communication avec le PC

L'interface la plus importante de la Raspberry est celle qui permet à l'utilisateur de manipuler les messages. Celle-ci se fait par l'intermédiaire du port série de la Raspberry. Les commandes accessible à l'utilisateur doivent être simple d'utilisation et les résultats doivent être facile à lire. Il est possible d'améliorer l'ergonomie du coté du PC en écrivant un programme qui s'occupera de mettre les données en forme. Cela signifie alors que l'utilisateur ne voit pas directement les données brutes à la sortie du port série.

Voici quelques contraintes utiles :

— **Une requête de l'utilisateur doit être traitée sans attente.** Cela implique qu'il faut écouter en permanence le port série et réagir à l'instant même ou la requête est reçu. Le résultat de certaines requête peut cependant prendre du temps (ex : envoi de message). Il faut alors faire comprendre à l'utilisateur que sa requête a bien été prise en compte (ex : un message `'sending...'`)
— **Les erreurs doivent être gérées.** À partir du moment ou l'on fait intervenir un humain, on peut s'attendre à n'importe quoi. Il faut donc que le programme soit robuste pour éviter de planter à la moindre faute de l'utilisateur
— **L'ensemble des requêtes définies dans le TP doivent être utilisable.** Vous avez le droit d'ajouter ou améliorer ces requêtes (ex : afficher les messages d'un destinataire précis). Ça ne doit cependant pas être une priorité.

2.5 Automatisation du système

Une fois que le système est opérationnel. Il peut être pratique d'automatiser le démarrage des programme lorsque la Raspberry est mise sous tension. Il faut pour cela utiliser les outils de systemd

et adapter les programmes pour qu'ils puissent fonctionner avec ces outils. De plus, la Raspberry étant un système embarqué, il peut être intéressant d'optimiser le démarrage pour qu'il se fasse le plus rapidement possible (ex : ne pas démarrer l'interface graphique).