

1 Le produit

1.1 Objectif

L'objectif de ce TP est de concevoir un *boîtier de messagerie* physique permettant à des utilisateurs de communiquer au moyen de messages textuels dès lors qu'il est relié à un ordinateur. Cette section décrit cet objet du point de vue d'un utilisateur.

1.2 Caractéristiques

Lorsque le boîtier est relié à un ordinateur, les actions suivantes sont possibles :

- s'enregistrer dans le réseau avec un nom
- envoyer un message à un destinataire
- lire les messages envoyés
- lire les messages reçus non lus
- lire les messages reçus déjà lus

1.3 Évènements

À l'initialisation (dès que l'on branche le boîtier à un ordinateur), le système n'a aucun message en attente et aucune action n'a été engagée par l'utilisateur, les voyants lumineux sont alors tous éteints (figure 1a).

Lorsqu'un message est reçu, le voyant vert (figure 1d) se met à clignoter. Il est éteint par l'évènement de lecture des messages reçus non lus.

Une action utilisateur telle que s'enregistrer dans le réseau ou envoyer un message déclenche une communication distante. Le voyant jaune (figure 1b) s'allume dès que l'action est prise en compte. Il est éteint lorsqu'une réponse a été reçue. Si la réponse informe d'une erreur, le voyant rouge s'allume jusqu'à la tentative suivante (figure 1c).



FIGURE 1 – Différents états du boîtier

2 Fiche technique

2.1 Configuration matérielle

Le boîtier sera conçu au moyen d'une Raspberry Pi disposant d'une extension comportant 3 LED¹, une verte, une rouge et une jaune (figure 2) et reliée à un ordinateur au moyen d'un port série (UART²). Pour le lien réseau avec les autres boîtiers, ce sera par Ethernet (figure 3).

Remarques : les Raspberry Pi doivent rester dans la salle de TP et être rangées dans l'armoire à la fin de chaque séance. Vous disposerez d'une carte SD (le *disque dur* de la Raspberry Pi) par binôme qui sera à rendre en fin de dernière séance.

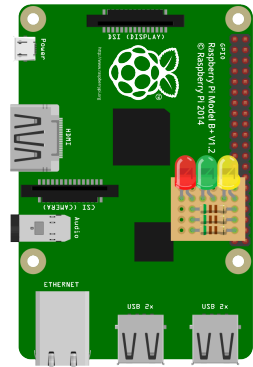


FIGURE 2

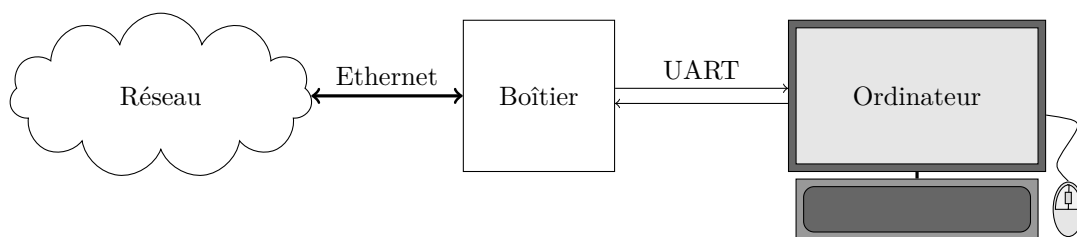


FIGURE 3 – Architecture du projet

L'extension doit être branchée sur une ligne de 4 broches comportant une masse (noire sur le schéma, figure 4) à une extrémité et 3 GPIO³ (jaune sur le schéma, figure 4).

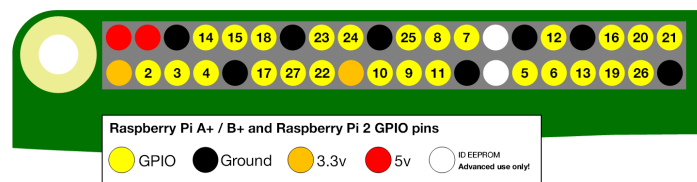


FIGURE 4 – Schéma des broches de la Raspberry Pi

2.2 Configuration logicielle

Le système d'exploitation exécuté sur la Raspberry Pi sera Raspbian⁴ (une distribution utilisant le noyau Linux basée sur Debian GNU/Linux). Vous devrez installer ce système durant la première séance de TP. Afin de pouvoir se servir du port série de la Raspberry Pi, il faudra configurer le système pour qu'il ne l'utilise pas. (voir <https://hallard.me/enable-serial-port-on-raspberry-pi/>)

1. Light-Emitting Diode
 2. Universal Asynchronous Receiver/Transmitter
 3. General Purpose Input/Output
 4. <https://www.raspbian.org/>

2.3 Protocole du serveur

Chaque message au serveur est de la forme suivante :

CMD [ARGUMENTS]\n

Le serveur supporte les commandes (CMD) suivantes :

nick pour se donner un nom

send pour envoyer un message

list pour lister les clients connectés

Enfin, le serveur peut envoyer au client des messages :

ack pour acquitter une commande reçue

recv pour faire parvenir un message envoyé

2.3.1 ack

Un message reçu du serveur avec la commande ack est toujours en réponse à un message envoyé au serveur. Ce message comporte un argument, numérique, correspondant à un code d'erreur de la commande à laquelle correspond cette réponse (les commandes étant traitées dans l'ordre de réception par le serveur). Les codes d'acquiescement non spécifiques à une commandes sont :

- | | |
|----------------------------|--|
| 0 (SUCCESS) | en cas de succès |
| 1 (UNKNOWN_COMMAND) | lorsque la commande n'est pas reconnue |

Pour les autres codes, voir les commandes concernées.

2.3.2 nick

Cette commande attend un argument : le nom.

Il doit respecter l'expression régulière suivante :

`^[a-zA-Z_][0-9a-zA-Z_]*$`

Remarque : le nom *all* est déjà pris par un client spécial qui permet d'envoyer à tous.

Les codes d'acquiescement qui peuvent être retournées par cette commande sont :

- | | |
|---------------------------------|---|
| 0 (SUCCESS) | en cas de succès |
| 2 (PARSE_ERROR) | s'il manque un argument |
| 3 (NICKNAME_UNAVAILABLE) | si le nom est déjà utilisé |
| 4 (INVALID_NICKNAME) | si le nom ne respecte pas le bon format |

2.3.3 send

Cette commande attend deux arguments : le destinataire et le message.

Le destinataire doit être le nom d'une personne existante (qui a déjà utilisé la commande **nick**).

Le message peut comporter des espaces, mais pas de retour à la ligne (puisqu'utilisé pour terminer la requête au serveur)

Les codes d'acquiescement qui peuvent être retournées par cette commande sont :

- | | |
|------------------------------|---|
| 0 (SUCCESS) | en cas de succès |
| 2 (PARSE_ERROR) | s'il manque un argument ou plus |
| 5 (NICKNAME_REQUIRED) | si l'expéditeur n'a pas de nom |
| 6 (UNKNOWN_RECIPIENT) | si le destinataire est inconnu |
| 7 (INVALID_MESSAGE) | si le message est invalide (par exemple vide) |

2.3.4 recv

Cette commande envoyée par le serveur ressemble à la commande **send**. Elle possède deux arguments, un expéditeur et le message.

L'expéditeur correspond au nom de la personne qui est source du message envoyé.

Comme pour la commande **send**, le message peut comporter des espaces.

2.3.5 list

Cette commande sert à récupérer la liste des clients connectés. Elle n'est pas faite pour être utilisée dans un programme mais plutôt manuellement avec **netcat** pour trouver facilement des clients avec qui communiquer.

3 Objectifs intermédiaires


Rappel : l'objectif est de concevoir le boîtier de messagerie présenté dans la partie 1. Pour y parvenir, plusieurs étapes sont proposées dans cette section.

3.1 Mise en place

3.1.1 Installation du système

Première étape du projet, installer un système d'exploitation sur la machine (la Raspberry Pi) : Raspbian⁵. Vous disposez pour cela d'une carte micro-SD, d'un adaptateur micro-SD vers USB pour y accéder depuis l'ordinateur à votre disposition.

 Commandes utiles : **man**, **lsblk**, **dd**

 Avant d'utiliser la commande **dd**, la faire valider par l'enseignant

3.1.2 Configuration du système

Il est ensuite nécessaire de configurer le système pour pouvoir :

- utiliser le port série
- sécuriser un minimum

Pouvoir utiliser le port série requiert de dire au système de ne pas l'utiliser, pour cela, dans la partition **boot**, il faut éditer le fichier **config.txt** pour y faire figurer la ligne suivante : **enable_uart=1**

Vous pouvez maintenant démarrer la Raspberry Pi (en la branchant).


Dès maintenant, il est important de vous créer un utilisateur (au moins) sur la Raspberry Pi, car toutes ont le même nom de compte (**pi**) et mot de passe **raspberrypi** par défaut. Vous pouvez ajouter cet utilisateur au groupe **sudo**, lui permettant d'utiliser la commande du même nom pour changer d'utilisateur (et donc devenir **root**).

 Commandes utiles : **man**, **adduser**

Si vous ne souhaitez pas utiliser la commande **sudo**, vous pouvez changer le mot de passe de l'utilisateur **root**.

 Commandes utiles : **man**, **passwd**

Enfin, n'oubliez pas de changer le mot de passe de l'utilisateur **pi**, ou de le supprimer du système.

 Commandes utiles : **man**, **passwd**, **deluser**

5. <https://www.raspbian.org/>

3.1.3 Mise en place du module

Vous pouvez brancher le module LED sur la Raspberry Pi comme indiqué dans la section « Fiche technique » (figure 4). Notez les numéros des GPIO sur lesquelles vous branchez les différentes LED, ils vous seront utiles par la suite pour les contrôler.


3.2 Création du projet

Créez votre projet, localement : un dossier qui sera versionné avec Git (voir section Git). Pour la sauvegarde distante, utilisez le GitLab de l'ISIMA (voir section Git > GitLab).

3.3 Pistes


Cette section est là pour vous guider sur des tests que vous pouvez faire :

- contrôler une LED en utilisant Linux en shell
- même chose en C
- faire clignoter une LED

 Rappels : `man, /sys/class/gpio` ^a


^a <http://elixir.free-electrons.com/linux/latest/source/Documentation/gpio/sysfs.rst>

- communiquer en UART avec le PC
- communiquer en TCP avec le serveur

 Nous fournissons quelques fonctions C pour vous aider sur ces parties

https://zz.cypierre.ovh/linux_emb/tp/

- créer un *daemon*
- le lancer et l'arrêter
- l'activer au démarrage (et vérifier)

 Le cours présente succinctement la conception de *daemon* avec `systemd`. Vous devriez avoir assez d'informations pour réussir, mais n'hésitez pas à utiliser `man`

4 Git

4.1 Git (local)

Cette section est destinée à ceux qui ne savent pas utiliser Git. Elle permet d'en avoir une utilisation basique, suffisante pour ces TP.

Git est un gestionnaire de version : il permet de sauvegarder votre code régulièrement, de revoir une version antérieure, ...

4.1.1 Commandes

Git possède de nombreuses commandes, mais les basiques sont :

config	permet de configurer certaines informations
init	active Git pour le répertoire et ses sous-répertoires
add	ajoute des modifications (fichiers) à valider
commit	valide des modifications
pull	télécharge des commits depuis un serveur
push	télécharge des commits vers un serveur

4.1.2 Configuration initiale

Avant de commencer, vous pouvez configurer votre système ainsi :

Pour la Raspberry Pi :

```
git config --global user.name "Prenom Nom"
git config --global user.email "prenom.nom@poste.isima.fr"
```

Pour le PC, ce sera les mêmes commandes mais avec les noms et adresse de l'autre utilisateur.

Dans le dossier principal de votre projet :

```
git init
```

4.1.3 Utilisation

Par la suite, lorsque vous voulez sauvegarder votre code dans l'état dans lequel il se trouve :

```
git add fichier1 fichier2 ...
git commit -m "message du commit"
```

Vous pouvez faire ces commandes très régulièrement. On le fait en général lorsque l'on a fait une étape dans son projet (une nouvelle fonction, par exemple). Plus c'est fait régulièrement, et plus une erreur, fausse manipulation (suppression d'un fichier, ...) est anodine, puisqu'il existe des sauvegardes.

4.2 GitLab (distant)

Enfin, vous pouvez sauvegarder votre code à distance, sur un serveur. L'ISIMA vous donne accès à un serveur : <https://gitlab.isima.fr>

GitLab est un outil web pour gérer la partie "serveur" de Git.

Vous pouvez vous identifier sur ce site avec vos identifiants habituels.


4.2.1 Configuration initiale

Afin d'éviter d'avoir à entrer son compte et mot de passe à chaque interaction avec le serveur, on va générer des clés SSH.

Sur la Raspberry Pi et le PC, vous pouvez générer une paire de clé :

```
ssh-keygen -t rsa
```

Vous trouverez dans `~/.ssh/id_rsa.pub` votre clé publique, qui doit être copiée.

 Sous linux, sélectionner du texte et le placer dans un presse-papier que l'on peut coller avec un clic du bouton central de la souris

Dans GitLab, vous avez un menu en haut à droite : Settings > SSH Keys

Collez dans le champ Key la clé publique, et validez (Add key)

Vous pouvez ajouter plusieurs clés (pour pouvoir y accéder depuis plusieurs ordinateurs)

4.2.2 Configuration du projet

Dans le dossier de votre projet (qui doit déjà être initialisé pour Git) :

```
git remote add origin git@gitlab.isima.fr:user/project.git
```

4.2.3 Sauvegarde et chargement

À la première sauvegarde, il faudra faire :

```
git push -u origin master
```

Par la suite :

```
git push
```

Pour récupérer les derniers commits :

```
git pull
```

5 Consignes de TP

5.1 À chaque fin de séance

- `git push`
- éteindre votre ordinateur
- éteindre l'écran
- ranger les câbles
- ranger la Raspberry Pi

5.2 À la fin de la dernière séance

- comme pour chaque fin de séance
- rendre la carte micro-SD