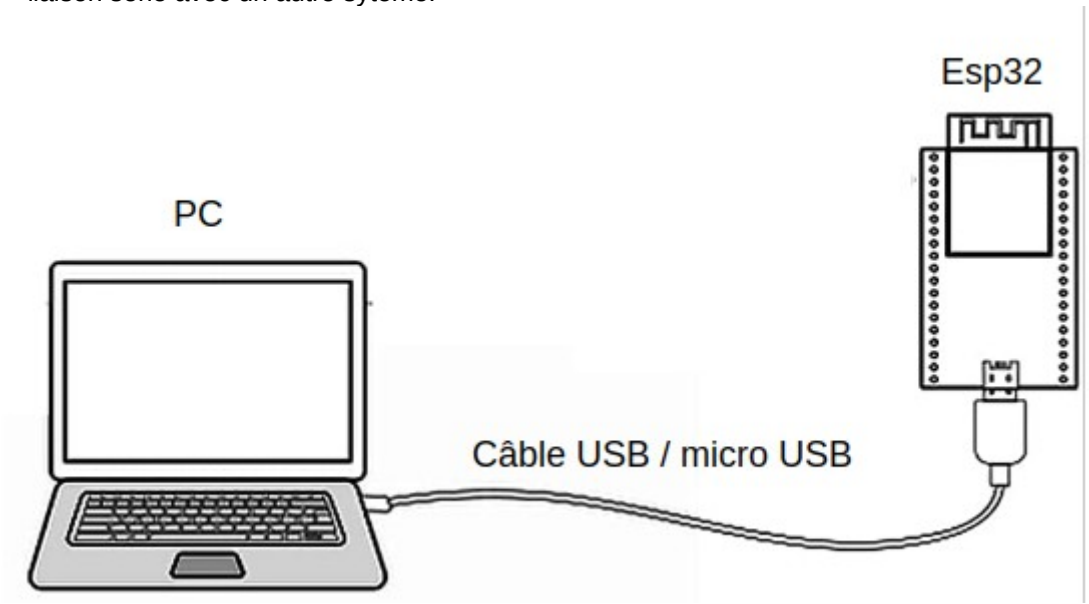
Communication entre systèmes

TD

1 Mise en œuvre de la liaison série avec esp32

Les ordinateurs types PC de bureau disposent de ports USB qui peuvent établir une liaison série avec un autre sytème.



2 Test de la liaison série sous Linux

Pour avoir des informations sur la liaison série, il suffit de tapez en ligne de commande :

```
stty -a < /dev/ttyUSB0
```

En vous aidant de la page de manuel de stty, donnez la signification des éléments suivants :

- speed
- parenb
- parodd
- ixon

Donnez la configuration actuelle de votre liaison série :

- Vitesse en baud :
- Nombre de bits par donnée :
- Nombre de bits de stop :
- Type de parité :
- Contrôle de flux :

Pour configurer les paramètres de la liaison série RS232, on peut utiliser également la commande stty :

```
stty -F /dev/ttyUSB0 vitesse "liste de drapeau"
```

Exemple :

```
stty -F /dev/ttyUSB0 9600 cs8 -parenb -ixon
```

La liaison série fonctionnera à 9600 bauds, chaque donnée sera codée sur 8 bits, avec activation de la parité, cette dernière étant impaire, il n'y aura pas de contrôle de flux xon/xoff (contrôle de flux logiciel).

Branchez un câble USB microUSB avec l' esp32.

Configurer votre port série afin qu'il possède les caractéristiques suivantes :

- Vitesse en baud : 9600
- Nombre de bits par donnée : 8
- Nombre de bits de stop : 1
- Type de parité : aucune
- Contrôle de flux : aucun (ni matériel, ni logiciel)

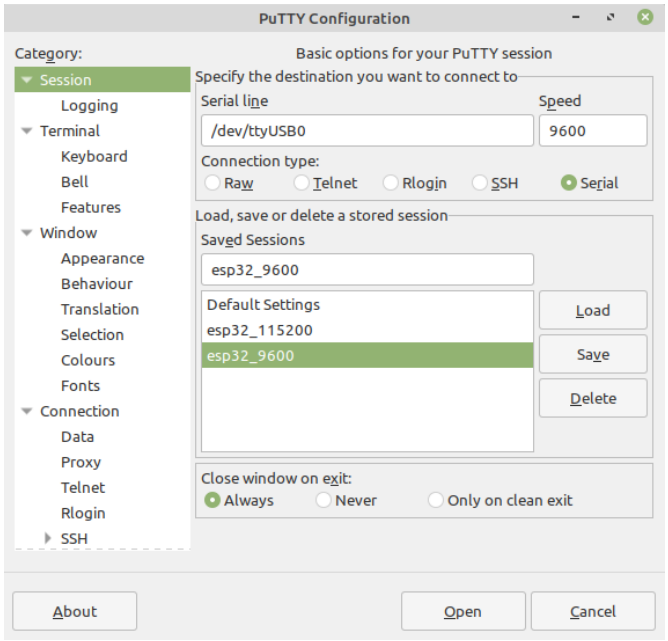
saisir la commande suivante : et appuyer sur le bouton reset de ESP32

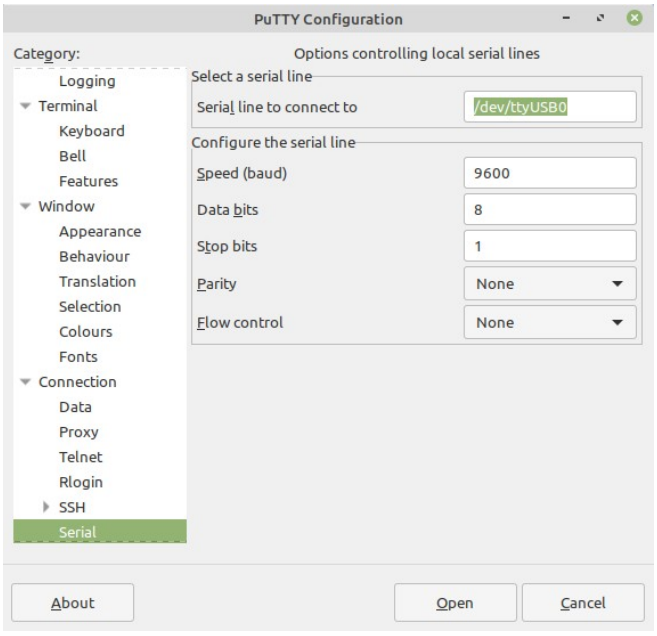
```
cat /dev/ttyUSB0
```

Quelle est la dernière phrase affichée ?

A l'aide de puTTY testez la communication avec l'esp32.

Le programme présent dans l'esp32 attend un caractère, suivant le caractère reçu (a,b,c …) le programme répondra en envoyant le texte d'une fable de La Fontaine ou autre.

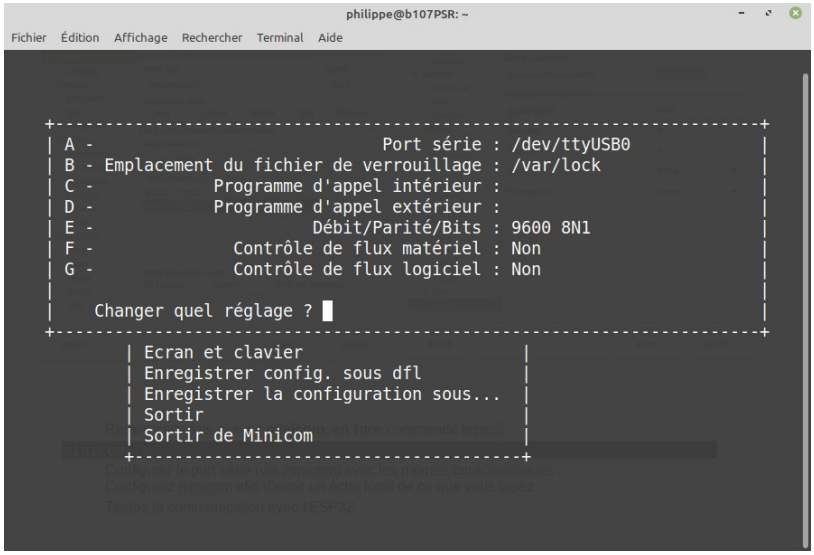




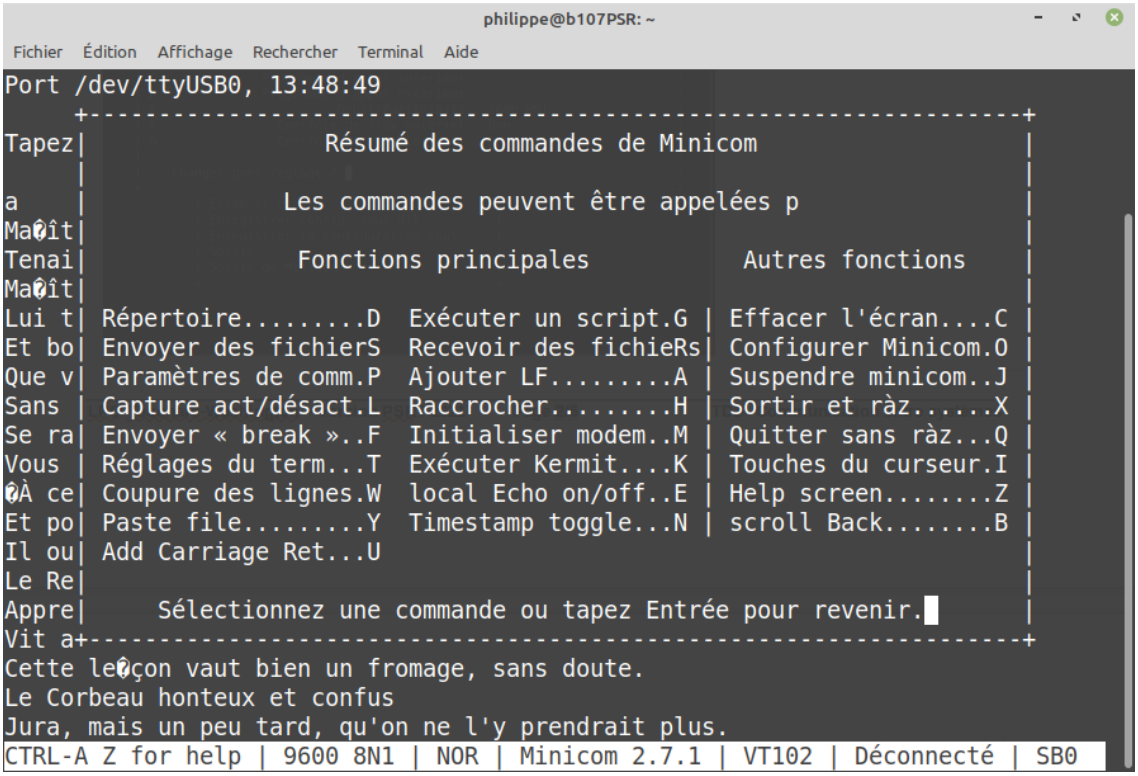
Refaire cette fois ci avec **minicom**, en ligne commande tapez :

```
minicom -s
```

Configurer le port série (via minicom) avec les mêmes caractéristiques .



Configurez minicom afin d'avoir un **écho local** (lettre E) de ce que vous tapez et afin d'avoir un **retour chariot** après chaque fin de ligne (lettre U)



Testez la communication, que constatez-vous ?

3 Programmation en langage C

1. Réalisez la fonction **OuvrirPort**, elle prend en paramètre d'entrée le nom du port série sous la forme d'une chaîne de caractères et retourne un descripteur de fichier (un entier).

2. En utilisant la librairie **termios** réalisez le code de la fonction **ConfigurerSerie**, elle prend en paramètre d'entrée la vitesse de transmission. Le code de cette fonction est donnée en annexe.

La structure **termios** de la librairie <sys/termios.h> permet de configurer la liaison série.

Elle est composée des champs suivants :

```
tcflag_t c_iflag;      /* modes d'entrée */
tcflag_t c_oflag;      /* modes de sortie */
tcflag_t c_cflag;      /* modes de contrôle */
tcflag_t c_lflag;      /* modes locaux */
cc_t     c_cc[NCCS];   /* caractères spéciaux */
```

Chaque champ peut prendre une ou plusieurs valeurs.

Pour assigner plusieurs valeurs à un champ, il suffit de les séparer par | (barre verticale OU binaire)

Pour proposer l'inverse d'une valeur, il faut la faire précédé du symbole ~ et faire un ET binaire.

Le code ci-dessous permet de fixer la taille des données à 7 bits, avec une vitesse de 9600 bauds et pas de parité :

```
struct termios tty ;
tty.c_cflag = B9600 | CS7
tty.c_cflag &= ~PARENB
```

3. Réalisez un programme principal qui ouvre le port série ttyUSB0 et le configure avec les paramètres suivant 9600 bauds 8bits par caractères. Ce programme attend un caractère saisi au clavier et l'envoie sur la liaison série**.** Lorsque la caractère z est reçu le programme ferme la liaison série et se termine.

4. Réalisez la fonction **recevoirMessage**, elle prend en paramètre d'entrée le descripteur de fichier, le caractère de fin de chaîne attendu et en paramètre de sortie un tableau pouvant contenir des caractères, la taille du tableau. Son rôle est de lire caractère par caractère le port série jusqu'à l'arrivée du caractère de fin. Les caractères ainsi reçus sont placés dans le tableau pour former une chaîne de caractères.

5. Compléter le programme principal permettant la lecture d'un message terminé par le caractère EOT (end of transmission) . Le message est affichée sur l'écran au fur et à mesure de sa réception.

Annexe 1 la fonction configuerSerie

```
void configurerSerie(const int fd, const int baud) {
    struct termios term;

    // La fonction tcgetattr permet d'obtenir les paramètres actuels d'une liaison.
    tcgetattr(fd, &term);

    /* c_iflag : les modes d'entrée
       Ils définissent un traitement à appliquer sur les caractères en provenance
       de la liaison série */
    term.c_iflag = IGNBRK | IGNPAR; // IGNBRK les signaux BREAK (ctrl Z) sont
ignorés.
    /* Inhibe le controle de flux XON/XOFF */
    term.c_iflag &= ~(IXON | IXOFF | IXANY);

    /* c_lflag : les modes locaux
       Il définit le mode (canonique ou non) et la gestion de l'écho.
       En mode canonique, les caractères reçus sont stockés dans un tampon
       et ils ne sont disponibles qu'à la réception d'un caractère EOL code ASCII
décimal 10.
       En mode non canonique les caractères sont disponibles immédiatement à la
lecture*/
    term.c_lflag = 0; // 0 mode non canonique

    /* c_oflag : les modes de sortie
       Ils définissent un traitement à appliquer sur les caractères envoyés sur la
liaison série.*/
    term.c_oflag = 0x00; // Pas de mode particulier

    /* c_cc : control characters
       VMIN : en mode non-canonique, spécifie le nombre de caractères
              que doit contenir le tampon pour être accessible à la lecture.
              En général, on fixe cette valeur à 1. */
    term.c_cc[VMIN] = 1;
    /* VTIME : en mode non-canonique, spécifie, en dixièmes de seconde,
               le temps au bout duquel un caractère devient accessible,
               même si le tampon ne contient pas c_cc[VMIN] caractères.
               Une valeur de 0 représente un temps infini. la lecture est donc
bloquante*/
    term.c_cc[VTIME] = 0;

    /* c_cflag : Les modes de contrôle
       Ce champ est important, car c'est ici que l'on définit le débit,
       la parité utilisée, les bits de donnée et de stop,
       8 bits de données, pas de parité */
    term.c_cflag &= ~(PARENB | CSIZE); // pas de parité
    term.c_cflag |= CS8; // 8 bits par caractère
    term.c_cflag |= CREAD; //ignore les signaux de contrôle modem

    /* vitesse de la transmission
       Vitesse de communication (version simple, utilisant les
       constantes  prédéfinies ) */

    speed_t myBaud;
    switch (baud) {
        case 50: myBaud = B50;
            break;
        case 75: myBaud = B75;
            break;
```

```
        case 110: myBaud = B110;
            break;
        case 134: myBaud = B134;
            break;
        case 150: myBaud = B150;
            break;
        case 200: myBaud = B200;
            break;
        case 300: myBaud = B300;
            break;
        case 600: myBaud = B600;
            break;
        case 1200: myBaud = B1200;
            break;
        case 1800: myBaud = B1800;
            break;
        case 2400: myBaud = B2400;
            break;
        case 4800: myBaud = B4800;
            break;
        case 9600: myBaud = B9600;
            break;
        case 19200: myBaud = B19200;
            break;
        case 38400: myBaud = B38400;
            break;
        case 57600: myBaud = B57600;
            break;
        case 115200: myBaud = B115200;
            break;
        case 230400: myBaud = B230400;
            break;
        default:
            myBaud = B9600;
    }

    if (cfsetispeed(&term, myBaud) < 0 || cfsetospeed(&term, myBaud) < 0) {
        printf("Erreur configuration\n");
    }

    /* Une fois que Les champs ont été modifiés,
       il faut enregistrer Les modifications au moyen
       de la fonction tcsetattr */
    tcsetattr(fd, TCSANOW, &term);

}
```