TD: Gestion des signaux Unix

Echauffement. . . .

Pour ce premier exercice d'échauffement, nous vous proposons de modifier le programme `C` suivant afin de capturer le signal d'interruption `SIGINT` et attendre 3 fois 'Ctr^C' avant de terminer le processus.

```c
static int rester = 1;
int main(int argc, char **argv)
{
   while (rester)
    {
    printf("Je suis toujours la!\n");
    pause(); // pause permet d attendre les signaux
    }
    exit(0);
}
```

Code Morse avec signaux Unix

Le Morse est un protocole de communication qui permet de transmettre des messages textes en utilisant un alphabet très simple codé avec une série d'impulsions. Nous pouvons utiliser différentes modalités pour représenter les impulsions comme le son, la lumière ou les gestes.

Historiquement, le Morse a été utilisé pour transmettre des télégrammes et des correspondances urgentes ou confidentielles. Il reste toujours utilisé jusqu'à nos jours pour transmettre des messages automatiques ou des messages urgents.

Nous proposons dans ce TD de réaliser une transmission de messages en code Morse en utilisant les signaux Unix comme moyen de représenter les impulsions. Ainsi, vous serez capables de transmettre un message texte entre deux processus en utilisant uniquement des signaux!

Principe du langage Morse

Le langage Morse est composé de deux lettres et deux séparateurs:

- Les deux lettres sont notées avec un point '.' et un tiret '-'
- Le premier séparateur permet de séparer deux lettres du message à transmettre; nous allons noter ce séparateur avec un tildé `~`
- Le deuxième séparateur permet de distinguer deux mots du message à transmettre; nous allons noter ce séparateur avec un espace ' '

La combinaison des deux lettres de Morse (le point et le tiret) va permettre de coder les 26 lettres de l'alphabet.

Voici quelques exemples:

- la lettre 'a' aura comme code : `.-`
- la lettre 'h' aura comme code : `....`
- la lettre 'e' aura comme code : `.`
- la lettre 'l' aura comme code : `.-..`
- la lettre 'o' aura comme code : `---`
- la lettre 'm' aura comme code : `--`

Voici à titre d'illustration, le codage en Morse du message suivant: 'hello me'

Le code Morse sera : `....~.~.-..~.-..~---  --~.`

Les signaux du Morse

Deux signaux sont nécessaires pour coder l'alphabet Morse avant la transmission sur un médium. Nous allons identifier ces signaux comme un *signal bas* et un *signal haut*. Le signal haut représente une impulsion sur un intervalle de temps et le signal bas représente l'absence de cette impulsion pendant le même intervalle de temps.

Dans la suite, les signaux bas et haut seront notés respectivement avec un 0 (silence) et 1 (impulsion)

La traduction de l'alphabet vers les signaux Morse est la suivante:

- Le point `.` sera codé avec un seul signal haut suivi d'un signal bas: 10
- Le tiret `-` sera codé avec 3 signaux hauts successifs suivis d'un signal bas : 1110
- le séparateur de lettres `~` sera codé avec 3 signaux bas successifs: 000
- le séparateur de mots ' ' sera codé avec 7 signaux bas successifs: 0000000

Par exemple, pour transmettre la lettre 'a' qui a pour code Morse `.-` nous utilisons les signaux : `101110`

Le message 'hello me' qui avait pour code Morse: `....~.~.-..~.-..~---  --~.` devient avec les signaux:

`10101010 000 10 000 1011101010 000 1011101010 000 111011101110 0000000 11101110 000 10 0000000` (les espaces ici sont utilisés pour améliorer la lisibilité)

Transmission de messages avec les signaux Unix

Pour établir un canal de communication entre deux processus, nous allons détourner l'usage habituel des signaux Unix pour transmettre des messages en Morse!

La librairie `signal.h` nous offre deux valeurs `SIGUSR1` et `SIGUSR2` qui sont libres d'utilisation. Nous allons utiliser `SIGUSR1` pour représenter un silence et `SIGUSR2` pour représenter une impulsion.

La communication sera unidirectionnelle et va s'établir entre un processus émetteur et un processus récepteur. Le processus émetteur devra connaître le PID du processus récepteur pour envoyer les signaux Unix.

Pour terminer correctement la communication le processus émetteur envoie le signal `SIGHUP` au récepteur.

Par exemple, pour transmettre la lettre `a` qui a pour code Morse `.-`, le processus émetteur devra envoyer la séquence de signaux suivante (de gauche à droite):

- SIGUSR2 SIGUSR1 SIGUSR2 SIGUSR2 SIGUSR2 SIGUSR1

Etape 1: Comprendre le code de l'émetteur

Dans un premier temps, téléchargez le code proposé avec le TD et consultez ensuite le code source des fichiers suivants `morse.h`, `utile.c` et `emetteur.c`

Lisez attentivement le code de la fonction:

```
void envoyer_message(const char *msg);
```

Etape 2: Mise en place des gestionnaires de signaux

Un premier code du serveur se trouve dans le fichier `morse.c`

Dans cette étape, nous vous demandons de simplement capturer les signaux `SIGUSR1`, `SIGUSR2` et `SIGHUP` envoyés par l'émetteur. Votre traitement se résumera à un simple affichage pour vous assurer de la bonne réception.

Essayez votre code en utilisant le programme émetteur.

Pour information, les noms des executables générés par le Malefile sont `emetteur` et `recepteur`.

Etape 3: Décodage des signaux

Complétez le code du serveur de l'étape précédente pour décoder les lettres et les mots et retrouver le message complet envoyé par l'émetteur.

Vous pouvez utiliser les fonctions utilitaires présentes dans le fichier `utile.c`

Etape 4 (Option): Gestion des exceptions et fin de communication avec setjmp

Modifiez le code du serveur pour gérer le signal de fin transmission et les erreurs en utilisant les mécanismes `setjmp/longjmp`