Système Unix

TP my_select

B-PSU-050 b-psu-050@epitech.eu

Abstract:
Ce TP a pour objectif de vous présenter le fonctionnement général des termcaps sous Unix.
A l'issue de ce TP, vous devriez être en mesure d'utiliser les subtilités disponibles dans le drivers des pseudo-tty, maitrisant ainsi les affichages "avancés" dans un ***xterm****.*

Enfin, ce TP a pour vocation de vous préparer au mini-projet associé : le ***my_select***

Table des matières

.1 Introduction

Le cours vous a sommairement présenté `/dev` et la notion de `device`.
Les fichiers contenus dans `/dev` (les devices donc) correspondent en fait à une liaison directe avec les drivers contenus dans le système (le noyau Unix).

Par exemple, les devices /dev/cdrom* correspondent aux devices permettant d'acceder au driver de chacun des lecteurs CD/DVD de votre ordinateur.

Les fonctions **open**, **read**, **write** que vous avez maintenant l'habitude d'utiliser sur des fichiers vont aussi fonctionner sur les devices.

Rappelez vous, sous Unix, tout est fichier!!! Il convient donc d'accéder à ''tout'' de la même façon (fichier, device, ...).

Cela va permettre d'échanger des informations avec le lecteur de CD/DVD.

Cependant, nous allons être amenés à réaliser plus que de simples écritures et lectures d'informations avec notre device. En effet, nous aimerions également pouvoir le paramétrer. Pour cela, notre nouvel ami **ioctl** (pour Input Output Control) a pour rôle de modifier les paramètres de notre driver afin de modifier le comportement de notre matériel.

En effet, ce n'est pas en lisant ou en écrivant sur un DVD que l'on va effectuer une éjection du disque, mais plutôt en demandant au driver de déclencher le mécanisme d'ouverture de la trappe du lecteur.

Une partie des devices correspond aux périphériques physiques, comme les disques durs, les ports série/parallèle/usb, les lecteurs, les périphériques d'entrée (souris, clavier), ...

L'autre partie correspond à des "pseudo-devices", sans relation avec du matériel présent dans l'ordinateur, mais voulant se comporter comme tel.

Les terminaux série (`/dev/ttyXY`) font partie de la première catégorie. **ioctl** permet alors de paramétrer la vitesse de transmission, le mode de buffering interne du système, l'affichage à l'écran de ce qui est tapé au clavier, ...
Les pseudo-terminaux (`/dev/ptyXY`) sont dans la deuxième catégorie. **ioctl** permettra dans ce cas de réaliser des paramétrages similaires aux terminaux série.

En effet, les pseudo-terminaux sont sensés simuler virtuellement le comportement de terminaux série ‘‘physiques’’. Il est donc logique que l’on puisse effectuer un paramétrage similaire.

Après cette longue introduction théorique, passons donc sans plus attendre à l’étape 1 !

.2 Etape 1 : tc[gs]etattr

Comme vous pouvez vous en douter, nous n'allons pas utiliser directement **ioctl** pour interagir avec le terminal.
En effet, votre système met à votre disposition des fonctions adaptées au contrôle des terminaux.

Celles que nous aborderons au sein de ces exercices sont **tcgetattr** et **tcsetattr** (pour Terminal Control Get/Set Attribute).

1. Exercice 1 :
   Pour cette première étape, il s'agit de modifier le comportement général de votre terminal.
   Vous avez sûrement remarqué que, par défaut, les terminaux "bufferisent" les données que vous saisissez et ne les transmettent au programme que lorsque vous appuyez sur la touche `Enter`.
   Pour mettre cela en évidence, réalisez un programme `get_key` qui :

   (a) Lit sur l'entrée standard ;

   (b) Affiche ce qui est lu ;

   (c) Recommence indéfiniment.

   Vous devriez constater que ce que vous tapez est bien transmis ligne par ligne.

   Il s'agit donc de désactiver ce comportement en faisant en sorte que le terminal transmette directement les caractères saisis au programme, sans la moindre bufferisation.
   Utilisez **tcgetattr** et **tcsetattr** pour modifier votre programme `get_key`. Faites en sorte que votre programme affiche la valeur ASCII de chaque caractère au fur et à mesure que les touches sont tapées.

`A quoi peut bien correspondre le filedescriptor (fd) passé en paramètre aux fonctions tcgetattr et tcsetattr ?`

2. Exercice 2 :
   Vous devriez être dans la situation suivante : à chaque touche tapée, le caractère s'affiche à l'écran, suivi de sa valeur ASCII.

Pourquoi le caractère s'affiche-t-il à l'écran alors que votre programme ne l'a pas demandé ?
En fait, par défaut, le driver du pseudo-tty est paramétré pour afficher immédiatement le caractère frappé.

Dans ce second exercice, vous devez réaliser un programme `get_passwd` qui propose de taper un mot de passe, et qui, pour chaque caractère tapé, n'affichera que le caractère '*'.
Une fois que l'on appuie sur `Enter`, le programme affichera le mot de passe tapé.

```
1 (gdx@tls)./get_passwd
2 Type your password : *****
3 Your password is : gizmo
4 (gdx@tls)
```

Comment parle-t-on à un driver déjà ?

.3 Etape 2 : tget*

Au cours de cette seconde étape, nous allons aborder l'étude de nouvelles fonctions (**tgetent**, **tgetstr**, **tgetnum**, ...) permettant, indirectement, de communiquer des ordres à notre terminal sans passer par le driver.

En effet, la majorité de ces fonctions ne dialoguent pas avec le driver, ni avec le terminal. Elles sont en fait de simples fonctions de recherche dans une "base de données" de capacités supportées par notre terminal.

On parle alors de **termcaps**, pour Terminal Capabilities.

En précisant le type de terminal utilisé et l'action que vous souhaitez réaliser, ces fonctions vont vous fournir une chaîne de caractères contenant des codes de contrôle compréhensibles par le terminal.
Il suffit alors de lui transmettre cette chaîne de caractères pour effectuer l'action désirée (effacer l'écran, déplacer le curseur, changer la couleur, ...).

Ces codes de contrôle sont reconnus par le terminal, en aucun cas par le driver du tty. Il n'y a donc pas d'utilisation d'**ioctl**.

Comment transmettre la chaîne de caractères au terminal ?
Rien de plus simple : **write**, **my_printf**, **my_putstr**.

Vous pouvez également jeter un oeil à la fonction **tputs** ...

1. Exercice 1 :
   Ecrire un programme `clear_screen` qui videra le contenu affiché de votre terminal.

   (a) Chargez la "base de données" des capacités de votre terminal en cours d'utilisation ;

(b) Trouvez le nom de la capacité correspondant à l'effacement de l'écran ;

(c) Récupérez la séquence correspondante et envoyez la au terminal ;

(d) Testez votre programme.

`man curs_termcap`

`A noter que c'est exactement ce que fait la commande clear de votre système.`

2. Exercice 2 :
   Ecrire un programme `rev_video` qui doit :

   - Prendre en paramètre une chaîne de caractères
   - Afficher le paramètre en mode vidéo inverse

   Là encore il s'agit de :

   (a) Repérer les noms des capacités ;

   (b) Récupérer les séquences de contrôle ;

   (c) Les envoyer au terminal.

3. Exercice 3 :
   Afin d'être définitivement prêt pour le `my_select`, il convient de savoir déplacer le curseur dans le terminal.
   Vous aurez pour cela besoin de la fonction **tgoto** en plus des autres fonctions.
   Plusieurs capacités existent, pour faire avancer, reculer, monter ... voire déplacer directement le curseur à des coordonnées précises. A vous de choisir celle(s) qui vous semble(nt) les plus adéquate(s).
   Réaliser un programme `counter` qui affiche un compteur à l'écran :

   - Le chiffre "1" s'affiche à l'écran
   - Le programme attend 1 seconde
   - Le chiffre "2" s'affiche à l'écran en remplacement du "1" (au même endroit)
   - Le programme attend 1 seconde

- Le chiffre “3” s’affiche à l’écran en remplacement du “2” (au même endroit)
- Le programme attend 1 seconde
- Le chiffre “4” s’affiche à l’écran en remplacement du “3” (au même endroit)
- ...

Merci d’avoir suivi ce TP jusqu’au bout. Vous devriez être fin prêt pour aborder le projet `my_select`.
N’hésitez pas à vous montrer curieux et à découvrir les nombreuses capacités de votre terminal afin d’enrichir votre projet.