**Mini-projet système : développement d'un noyau de système d'exploitation**

Responsable : Christophe Rippert
`Christophe.Rippert@Grenoble-INP.fr`

# Gestion de l'écran en mode CGA

## Introduction

Dans ce mini-projet, on va se limiter à un mode d'affichage très simple géré par toutes les cartes graphiques depuis le début des années 80 (norme CGA, pour *Color Graphics Adapter* : la première carte vidéo gérant les couleurs sur les PC compatibles IBM).

## Spécification de l'affichage à l'écran

L'écran que nous considérons est le mode texte de base des cartes vidéo des PC dans lequel le noyau démarre. L'affichage fait 80 colonnes sur 25 lignes : les lignes et colonnes sont traditionnellement numérotées à partir de 0 (donc de 0 à 24 pour les lignes et de 0 à 79 pour les colonnes). L'affichage s'effectue en écrivant directement dans la mémoire vidéo pour y placer les caractères et leur couleur. Certaines opérations simples d'entrées-sorties sont nécessaires pour déplacer le curseur clignotant qui indique la position actuelle d'affichage.

### Principe

L'écran est couplé à une zone de la mémoire vidéo commençant à une adresse dépendant du mode utilisé (ici l'adresse de début est 0xB8000) : tout ce qui est écrit dans cette zone mémoire est donc immédiatement affiché à l'écran. Dans le mode vidéo utilisé, l'écran peut être vu comme un tableau de $80 \times 25 = 2000$ cases. Chaque case représente un caractère affiché à l'écran, et est composée de 2 octets (un octet = `uint8_t` en C) :

- le premier octet contient le code ASCII du caractère ;
- le deuxième octet contient le format du caractère, c'est à dire la couleur du texte, la couleur du fond et un bit indiquant si le texte doit clignoter, comme détaillé ci-dessous :

| **bit 7** | **bits 6, 5 et 4** | **bits 3, 2, 1 et 0** |
|---|---|---|
| clignote | couleur du fond | couleur du texte |

Attention : le clignotement n'est pas géré correctement par l'environnement d'exécution, vous devez forcer le bit 7 toujours à 0.
Les couleurs disponibles sont listées ci-dessous :

| **valeur** | **couleur** | **valeur** | **couleur** | **valeur** | **couleur** | **valeur** | **couleur** |
|---|---|---|---|---|---|---|---|
| 0 | noir | 4 | rouge | 8 | gris foncé | 12 | rouge clair |
| 1 | bleu | 5 | magenta | 9 | bleu clair | 13 | magenta clair |
| 2 | vert | 6 | marron | 10 | vert clair | 14 | jaune |
| 3 | cyan | 7 | gris | 11 | cyan clair | 15 | blanc |

Les 16 couleurs sont possibles pour le texte, par contre seules les 8 premières peuvent être sélectionnées pour le fond.
Pour afficher un caractère à la ligne `lig` et à la colonne `col` de l'écran, on doit donc écrire dans le mot de 2 octets (`uint16_t` en C) dont l'adresse en mémoire peut être calculé à partir de la simple formule suivante : $0xB8000 + 2 \times (lig \times 80 + col)$.

## Gestion du curseur

Lorsqu'on écrit dans un terminal en mode texte, on voit s'afficher un curseur clignotant qui indique la prochaine case dans laquelle on va écrire. Dans le mode vidéo que l'on utilise, ce curseur est géré directement par la carte vidéo : il suffit de lui indiquer à quelles coordonnées elle doit l'afficher.
On communique pour cela via des ports d'entrée-sorties : il s'agit de canaux de communication reliant les périphériques et dont les adresses sont fixées. Il existe deux types de ports :

— les ports de commandes qui servent à indiquer au périphérique l'opération souhaitée ;
— les ports de données qui permettent de communiquer effectivement avec le périphérique, en lisant ou en envoyant des données.

Dans l'architecture x86, il existe 65536 ports : le numéro d'un port est donc une valeur sur 16 bits. Cependant, on ne peut pas accéder aux ports directement via des pointeurs : on doit utiliser des instructions particulières.
Il existe des instructions assembleur dédiées pour la communication via les ports : sur l'architecture x86, il s'agit de l'instruction `in` (pour lire une donnée en provenance d'un port et la stocker dans un registre du processeur) et de l'instruction `out` (pour envoyer une donnée à un port). Ces deux instructions s'écrivent d'une façon particulière :

— `inb %dx, %al` : lit un octet de donnée sur le port dont le numéro est dans `%dx` et le stocke dans `%al` : attention, on doit obligatoirement utiliser les registres `%al` et `%dx`, à l'exclusion de tout autre ;
— `outb %al, %dx` : envoie l'octet contenu dans `%al` sur le port dont le numéro est dans `%dx`.

Bien sûr, il existe des équivalents pour lire des valeurs sur plus de 8 bits (`inw`, `ouw`, ...) mais on ne les utilisera pas dans ce projet.
Pour éviter d'avoir à écrire des bouts de fonction en assembleur, on fournit dans la mini-bibliothèque C (fichier d'en-tête `cpu.h`) des fonctions qui appellent elles-même les instructions `in` et `out` :

— `uint8_t inb(uint16_t num_port)` : renvoie l'octet lu sur le port de numéro `num_port` ;
— `void outb(uint8_t val, uint16_t port)` : envoie la valeur `val` sur le port `num_port`.

Lorsque vous écrivez des fonctions C qui doivent faire des entrée-sorties sur les ports, vous devez utiliser ces fonctions et ne surtout pas essayer d'ajouter de l'assembleur directement dans votre code C (l'assembleur *inline* est très complexe à mettre au point).
Dans les cartes vidéos VGA que l'on utilise dans ce TP, le port de commande gérant la position du curseur est le 0x3D4 et le port de données associé est le 0x3D5. La position du curseur est un entier sur 16 bits calculé via la formule suivante : $pos = col + lig \times 80$. Cette position doit être envoyée en deux temps à la carte vidéo : la succession d'opérations à effecter est donc la suivante :

1. envoyer la commande 0x0F sur le port de commande pour indiquer à la carte que l'on va envoyer la partie basse de la position du curseur ;
2. envoyer cette partie basse sur le port de données ;
3. envoyer la commande 0x0E sur le port de commande pour signaler qu'on envoie maintenant la partie haute ;
4. envoyer la partie haute de la position sur le port de données.

## Les caractères à afficher

On considère dans ce TP les caractères de la table ASCII (`man ascii`), qui sont numérotés de 0 à 127 inclus. Les caractères dont le code est supérieur à 127 (accents, ...) seront ignorés.
Les caractères de code ASCII 32 à 126 doivent être affichés en les plaçant à la position actuelle du curseur clignotant et en déplaçant ce curseur sur la position suivante : à droite, ou au début de la ligne suivante si le curseur était sur la dernière colonne.
Les caractères de 0 à 31, ainsi que le caractère 127 sont des caractères de contrôle. Le tableau ci-dessous décrit ceux devant être gérés. Tous les autres caractères de contrôle doivent être ignorés.

| Code | Mnémonique | Syntaxe C | Effet |
|---|---|---|---|
| 8 | BS | `\b` | Recule le curseur d'une colonne (si colonne $\neq 0$) |
| 9 | HT | `\t` | Avance à la prochaine tabulation (colonnes 0, 8, 16, ..., 72, 79) |
| 10 | LF | `\n` | Déplace le curseur sur la ligne suivante, colonne 0 |
| 12 | FF | `\f` | Efface l'écran et place le curseur sur la colonne 0 de la ligne 0 |
| 13 | CR | `\r` | Déplace le curseur sur la ligne actuelle, colonne 0 |