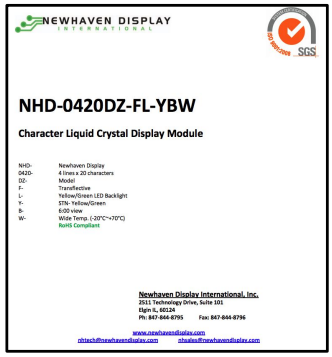
Pilote LCD

Module IOC — MU4IN109

Franck Wajsbürt

LCD

Présentation du LCD


NEWHAVEN DISPLAY INTERNATIONAL

SGS

NHD-0420DZ-FL-YBW

Character Liquid Crystal Display Module

NHD-	Newhaven Display
0420-	4 lines x 20 characters
DZ-	Model
F-	Transflective
L-	Yellow/Green LED Backlight
Y-	STN- Yellow/Green
B-	6:00 view
W-	Wide Temp. (-20°C~+70°C)

RoHS Compliant

Newhaven Display International, Inc.
2511 Technology Drive, Suite 101
Elgin IL, 60124
Ph: 847-844-8795 Fax: 847-844-8796

www.newhavendisplay.com
nhtech@newhavendisplay.com nhsales@newhavendisplay.com

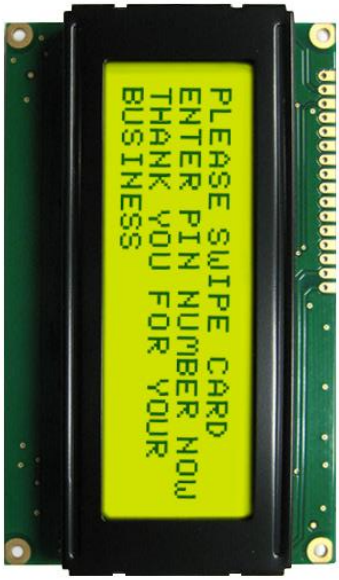
http://www.newhavendisplay.com/specs/NHD-0420DZ-FL-YBW.pdf

→ http://www.newhavendisplay.com/app_notes/ST7066U.pdf
http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

Dans une documentation technique, on trouve :

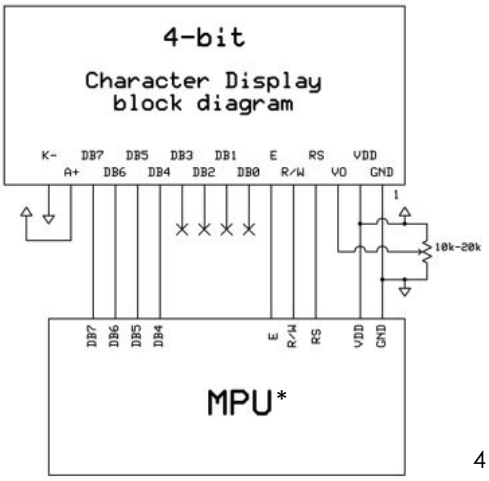
- La connectique
- Un schéma interne du ou des circuits de contrôle
- Les instructions interprétées par le micro-contrôleur interne
- Les chronogrammes pour l'envoi des commandes ou la lecture des données internes
- Les séquences spécifiques
  - Démarrage, arrêt, etc
- Des morceaux de codes d'usage

Connectique du LCD



Pin No.	Symbol	External Connection	Function Description
1	Vss	Power Supply	Ground
2	VDD	Power Supply	Supply voltage for logic (+5.0V)
3	V0	Power Supply	Power supply for contrast (approx. 0.5V)
4	RS	MPU	Register select signal. RS=0: Command, RS=1: Data
5	R/W	MPU	Read/Write select signal, R/W=1: Read R/W:=0: Write
6	E	MPU	Operation enable signal. Falling edge triggered.
7-10	DB0-DB3	MPU	Four low order bi-directional three-state data bus lines. These four are not used during 4-bit operation.
11-14	DB4-DB7	MPU	Four high order bi-directional three-state data bus lines.
15	LED+	Power Supply	Power supply for LED Backlight (+5.0V via on-board resistor)
16	LED-	Power Supply	Ground for backlight

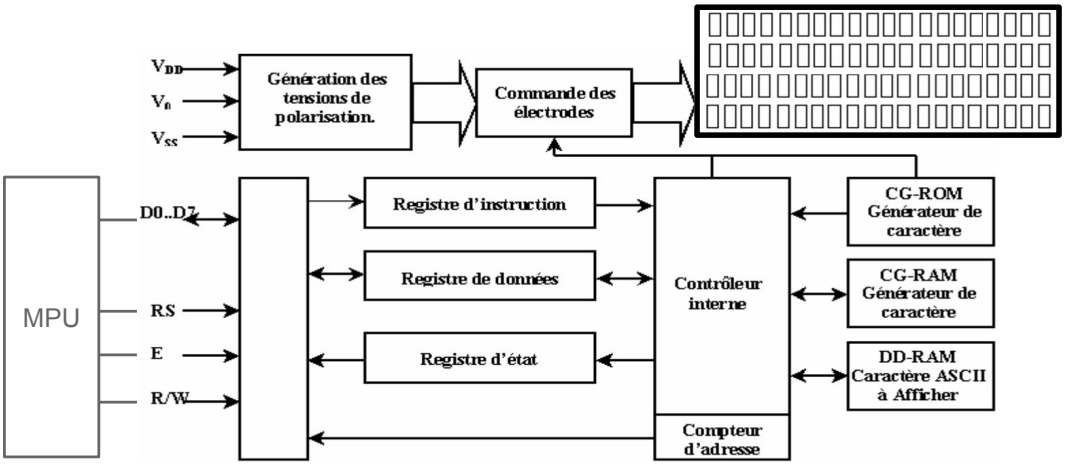
- 4 lignes de 20 caractères
- 2 modes : 4 bits et 8 bits
- Fonts programmables
- Mémoire interne en R/W



http://www.newhavendisplay.com/specs/NHD-0420DZ-FL-YBW.pdf

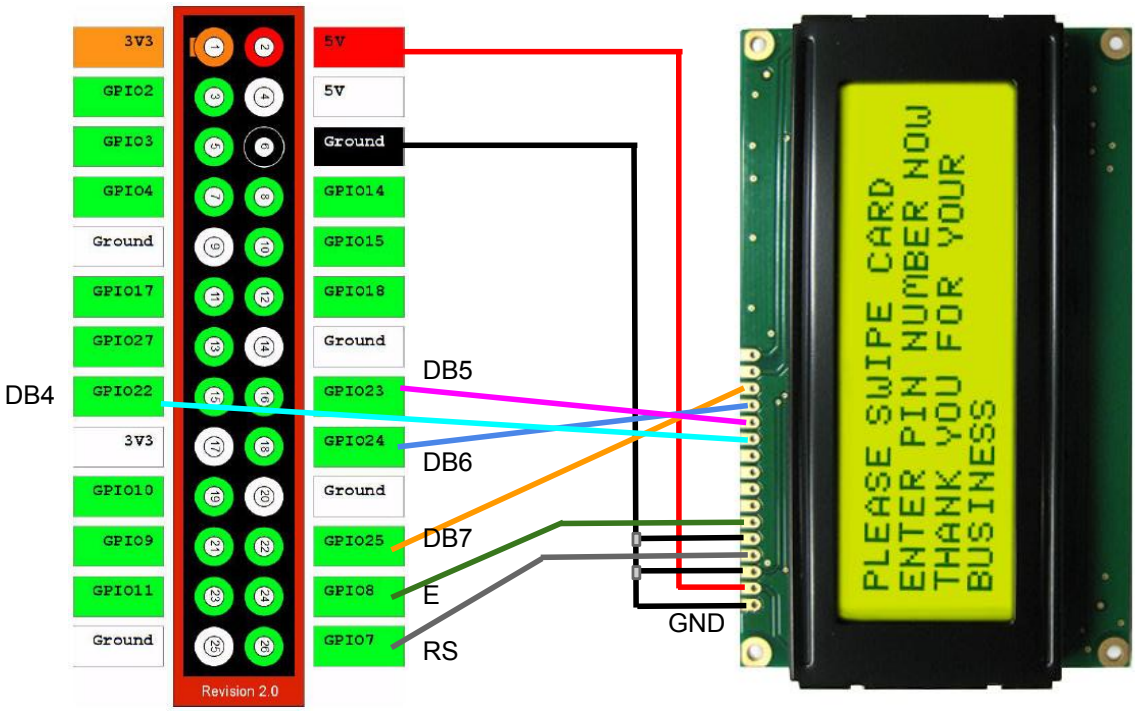
* MPU signifie Micro Processor Unit
ici c'est la Raspberry Pi

Contrôleur interne du LCD



- Le contrôleur du LCD est un ST7066U compatible avec le HD44780
  http://www.newhavendisplay.com/app_notes/ST7066U.pdf
  http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller
- C'est un ASIC dédié configurable (p. ex. Générateur de caractères)

Connexion avec la Raspberry Pi



Jeu d'instructions

HD44780U based instruction set

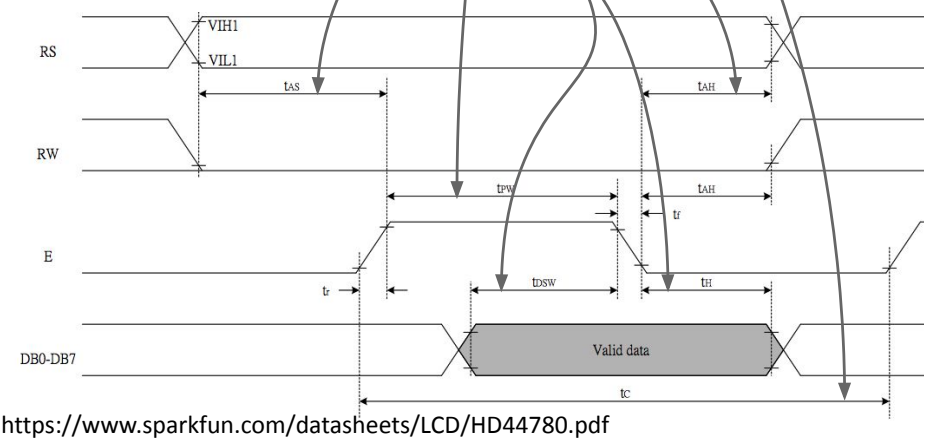
Instruction	Code										Description	Execution time (max) (when f_{cp} = 270 kHz)
	RS	R/W	B7	B6	B5	B4	B3	B2	B1	B0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).	1.52 ms
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position. Also returns display being shifted to the original position. DDRAM content remains unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D); specifies to shift the display (S). These operations are performed during data read/write.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display (D), cursor on/off (C), and blink of cursor position character (B).	37 µs
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM content remains unchanged.	37 µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.	37 µs
Read busy flag & address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).	0 µs
Write CGRAM or DDRAM	1	0	Write Data								Write data to CGRAM or DDRAM.	37 µs
Read from CG/DDRAM	1	1	Read Data								Read data from CGRAM or DDRAM.	37 µs

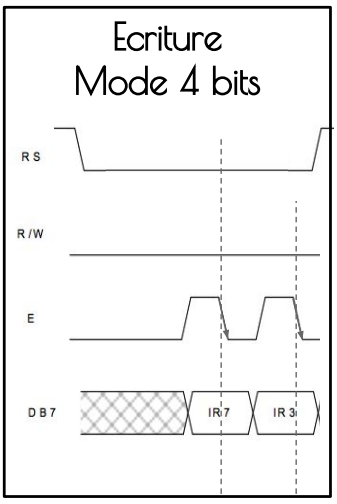
Instruction bit names —

I/D - 0 = decrement cursor position, 1 = increment cursor position; **S** - 0 = no display shift, 1 = display shift; **D** - 0 = display off, 1 = display on; **C** - 0 = cursor off, 1 = cursor on; **B** - 0 = cursor blink off, 1 = cursor blink on ; **S/C** - 0 = move cursor, 1 = shift display; **R/L** - 0 = shift left, 1 = shift right; **DL** - 0 = 4-bit interface, 1 = 8-bit interface; **N** - 0 = 1/8 or 1/11 duty (1 line), 1 = 1/16 duty (2 lines); **F** - 0 = 5×8 dots, 1 = 5×10 dots; **BF** - 0 = can accept instruction, 1 = internal operation in progress.

Séquence d'écriture

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	1000	—	—	ns
Enable pulse width (high level)	PW_{EH}	450	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	60	—	—	
Address hold time	t_{AH}	20	—	—	
Data set-up time	t_{DSW}	195	—	—	
Data hold time	t_H	10	—	—	





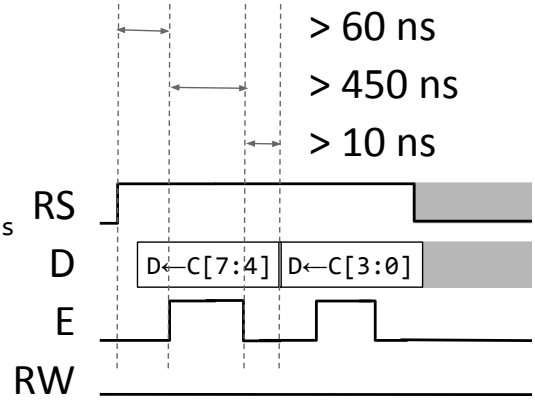
RS=0 commande
=1 data
R/W=0 write
E enable
DB data

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Commande par les GPIO

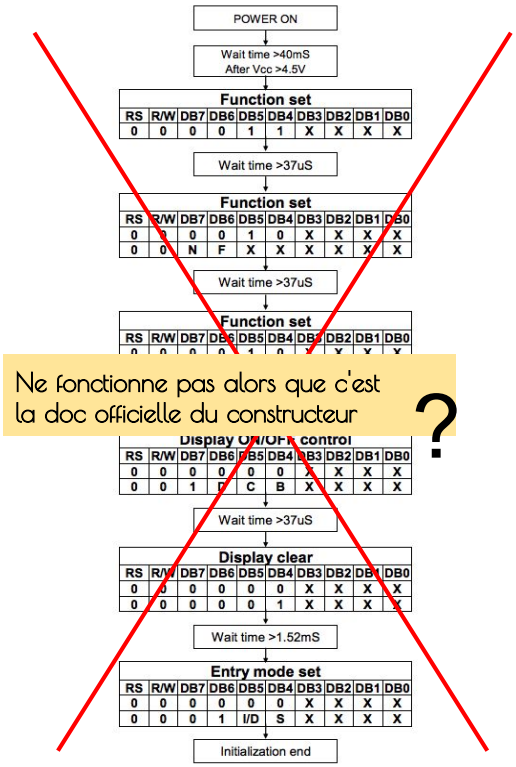
On ne peut pas reproduire précisément les chronogrammes
mais ce n'est pas grave, les durées à respecter sont des minimum

```
#define BIT(b,n)   (((n)>>(b))&1)
write_data ( char c ) {
     gpio_write (GPIO_RS, 1);
     gpio_write (GPIO_D7, BIT(7,c));
     gpio_write (GPIO_D6, BIT(6,c));   > 60ns
     gpio_write (GPIO_D5, BIT(5,c));
     gpio_write (GPIO_D4, BIT(4,c));
     gpio_write (GPIO_E,  1);          > 450ns
     usleep(1);
     gpio_write (GPIO_E,  0);          > 10ns
     gpio_write (GPIO_D7, BIT(3,c));
     gpio_write (GPIO_D6, BIT(2,c));
     gpio_write (GPIO_D5, BIT(1,c));
     gpio_write (GPIO_D4, BIT(0,c));
     gpio_write (GPIO_E,  1);
     usleep(1);
     gpio_write (GPIO_E,  0);
     usleep(50);
}
```




```
void gpio_write(int gpio, int value)
{
    int regnum = gpio / 32;
    int offset = gpio % 32;
    if (value)
        gpio_regs->gpset[regnum] = (0x1 << offset);
    else
        gpio_regs->gpclr[regnum] = (0x1 << offset);
}
```

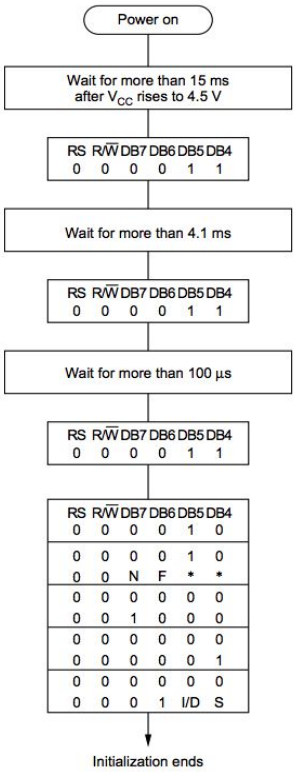

Initialisation



Ne fonctionne pas alors que c'est la doc officielle du constructeur ?

parfois la documentation officielle est fausse ....

mais on fini par trouver



http://www.newhavendisplay.com/app_notes/ST7066U.pdf https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

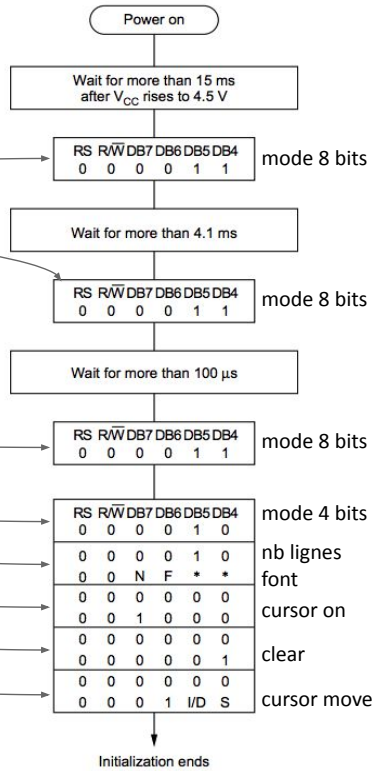
Initialisation

HD44780U based instruction set

Instruction	Code									Description
	RS	R/W	B7	B6	B5	B4	B3	B2	B1	B0	
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position. Also returns display being shifted to the original position. DDRAM content remains unchanged.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D), specifies to shift the display (S). These operations are performed during data read/write.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display (D), cursor on/off (C), and blink of cursor position character (B).
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM content remains unchanged.
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.
Read busy flag & address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).
Write CGRAM or DDRAM	1	0	Write Data								Write data to CGRAM or DDRAM.
Read from CG/DDRAM	1	1	Read Data								Read data from CGRAM or DDRAM.

Instruction bit names —

I/D - 0 = decrement cursor position, 1 = increment cursor position; **S** - 0 = no display shift, 1 = display shift; **D** - 0 = display off, 1 = display on; **C** - 0 = cursor off, 1 = cursor on; **B** - 0 = cursor blink
blink on ; **S/C** - 0 = move cursor, 1 = shift display; **R/L** - 0 = shift left, 1 = shift right; **DL** - 0 = 4-bit interface, 1 = 8-bit interface; **N** - 0 = 1/8 or 1/11 duty (1 line), 1 = 1/16 duty (2 lines); **F** - 0 = 5x8 d
dots; **BF** - 0 = can accept instruction, 1 = internal operation in progress.

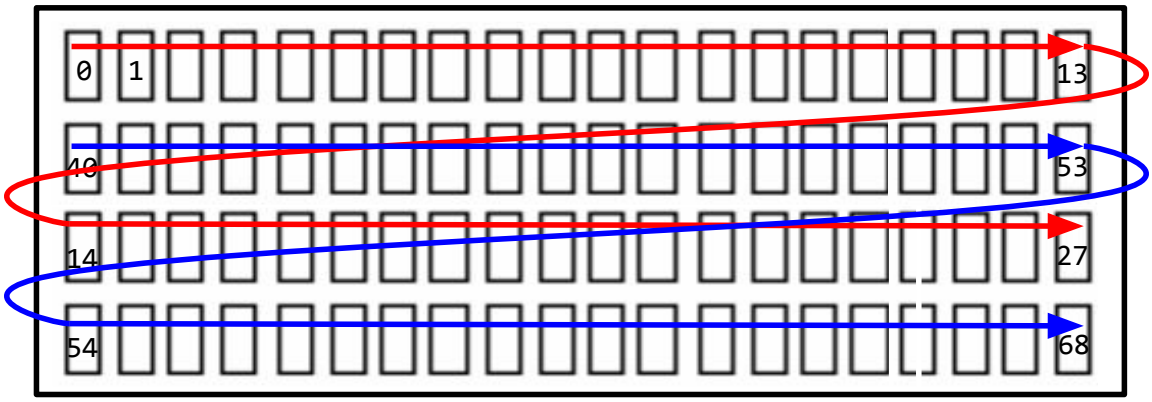


adressage des lignes

Cet afficheur ne contient que 2 lignes mais repliées

- Ligne 1 en rouge
- Ligne 2 en bleu

Les adresses, ici sont en hexadécimale 0x13 = 19, 0x40 = 64

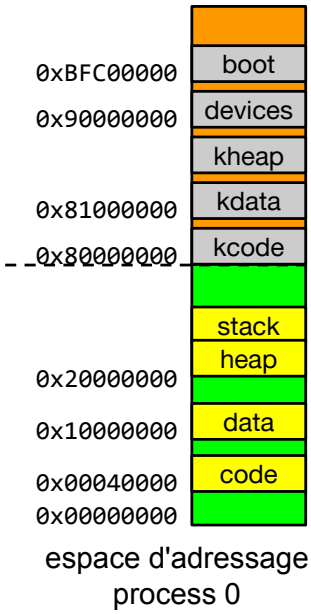


driver suite
kmalloc et ioctl

kmalloc

Espace d'adressage virtuel

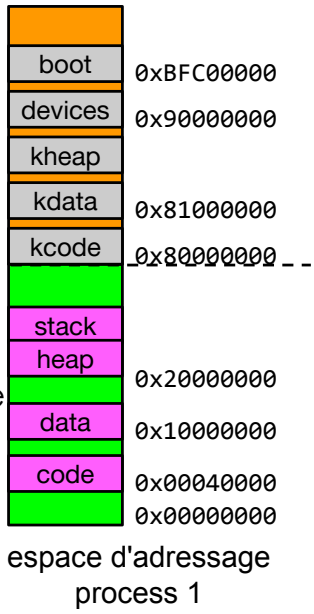
Chaque processus dispose d'un espace d'adressage propre.



Le processeur exécute le code du processus dans un espace virtuel, cela signifie que le processeur ne manipule que des adresses virtuelles.

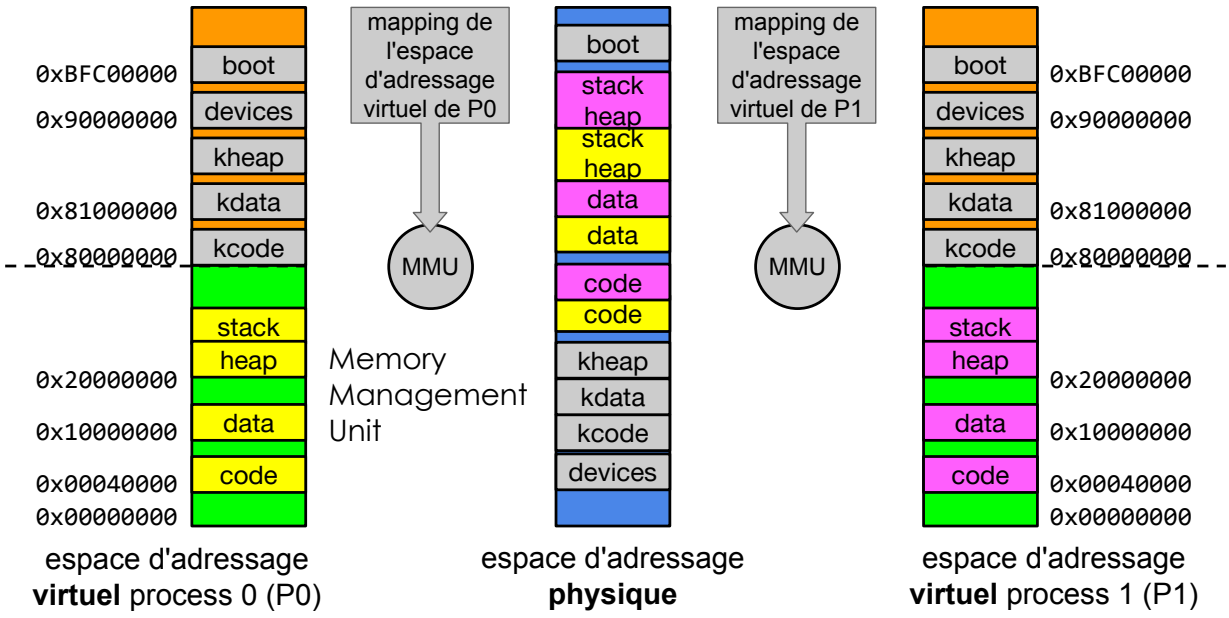
Le processeur ignore qu'il travaille dans un espace virtuel.

L'espace virtuel continue à être découpé en deux, une partie pour le noyau (accessible en mode kernel) et une partie pour l'utilisateur.



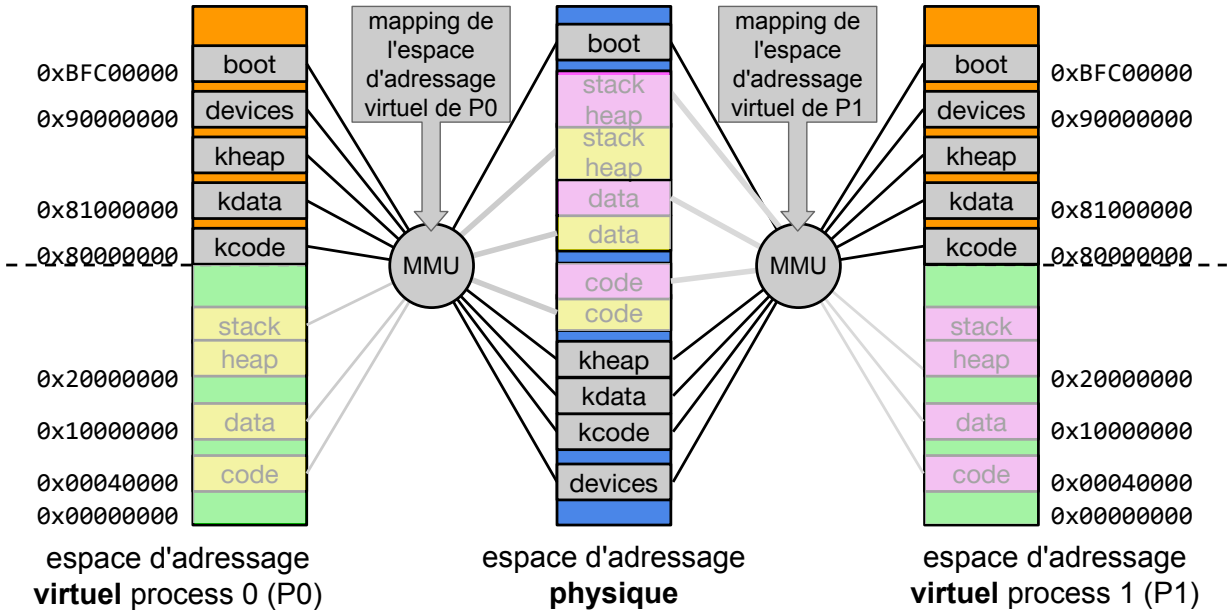
Mapping de l'espace virtuel

Les segments de l'espace d'adressage virtuel de chaque processus sont mappés dans des segments de l'espace d'adressage physique.
Ce sont les composants MMU qui se chargent de la traduction d'adresses



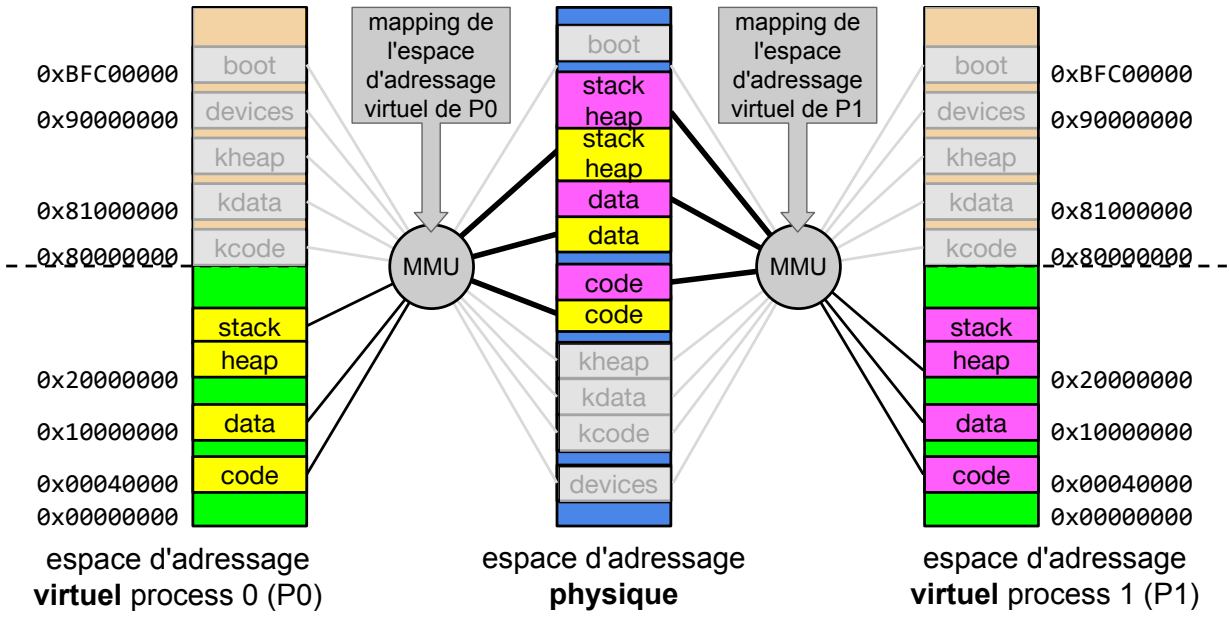
Mapping de l'espace virtuel noyau

Les segments virtuels utilisés par l'OS sont mappés dans les mêmes segments physiques pour tous les processus.



Mapping de l'espace virtuel utilisateur

Les segments de l'utilisateur sont évidemment mappés à des adresses différentes pour chaque processus.



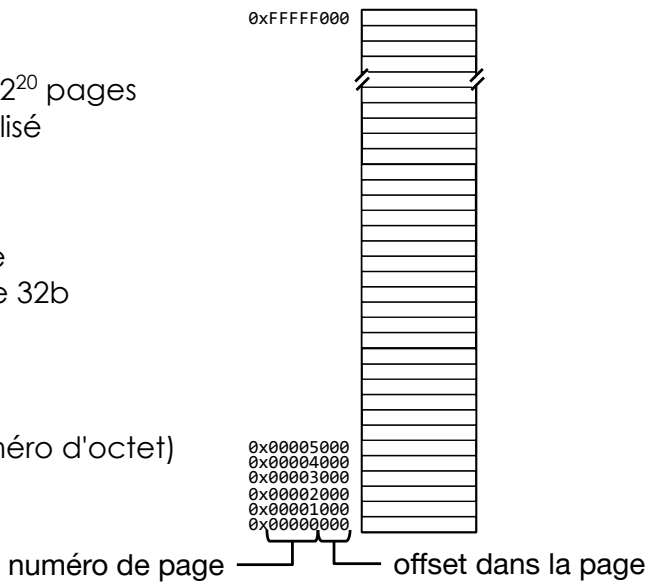
Mémoire paginée

Une page est un segment d'adresse, aligné en mémoire, de taille fixe.
Typiquement, la taille d'une page est de 4kB = 2^{12} Bytes

- L'espace d'adressage physique est découpé en pages
  Si une page fait 2^{12}Bytes alors il y a 2^{20} pages
- La page est l'élément atomique utilisé pour le mapping.

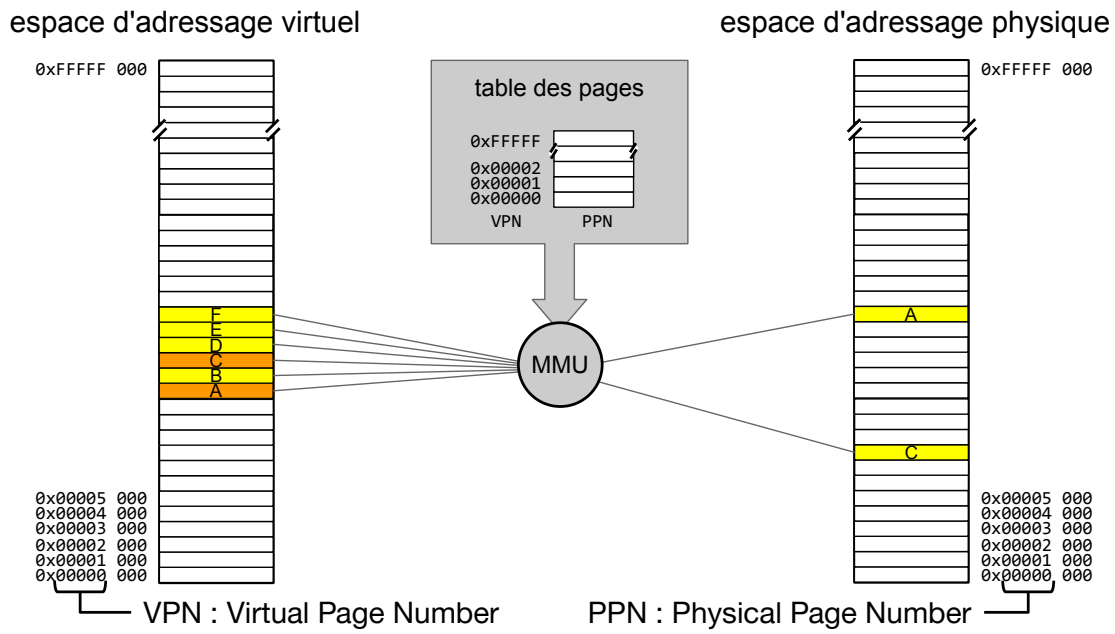
Le dessin représente le découpage en pages d'un espace d'adressage 32b

- Les 20 bits de poids forts sont le numéro de page
- Les 12 bits de poids faible sont l'offset dans la page (numéro d'octet)



Allocation des pages physiques

- Chaque segment d'adresse virtuel est aligné sur une page et sa taille est un nombre entier de page, chaque segment virtuel est formé d'un nombre entier de pages virtuelles.
- Le mapping consiste à mapper chaque page virtuelle dans une page physique.
- Le mapping est décrit dans une table des pages (tableau indexé par VPN contenant PPN)
- Un segment virtuel est mappé à la demande.



Translation Look-aside Buffer : TLB

Pour accélérer la traduction d'adresse, on place dans les MMU des caches de traductions d'adresses : les TLB.

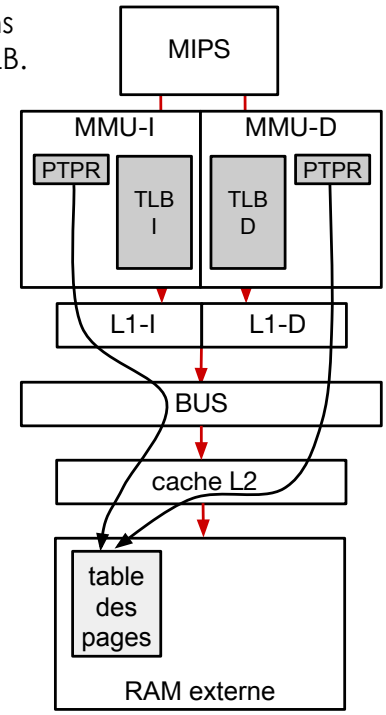
Les TLB sont des extraits de la table de page.

Ici, une TLB à 4 entrées contenant les 4 dernières traductions lues dans la table de pages.

1	VPN	PPN + flags
0		
0		
0		

Pour chaque entrée, il y a
un bit de validité et un couple VPN-PPN

Ce cache est petit (moins de 64 entrées)
et le plus souvent associatif (n'importe quelle case peut servir à n'importe quelle traduction).



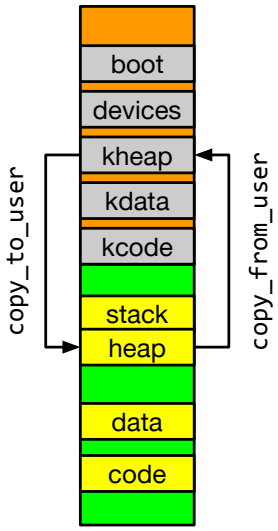
Allocation mémoire dans le noyau

En mode noyau, l'allocation mémoire utilise **`kmalloc`** et la désallocation **`kfree`**.
Ces fonctions ont un argument de priorité supplémentaire pour la fonction **`kmalloc()`**

- **GFP_KERNEL** : allocation normale de la mémoire du noyau
- **GFP_USER** : allocation mémoire pour le compte utilisateur (faible priorité)
- **GFP_ATOMIC** : alloue la mémoire à partir du gestionnaire d'interruptions

```
#include <linux/slab.h>
buffer = kmalloc(64, GFP_KERNEL);
if(buffer == NULL) {
    printk(KERN_WARNING "problème kmalloc !\n");
    return -ENONEM;
}
kfree(buffer), buffer = NULL;     // l'assignation de buffer évite
                                  // l'usage après la désallocation
```

copie entre espace d'adressage kernel et user



Les opérations de type `read()` ou `write()` du device passent un pointeur sur le buffer dans l'espace utilisateur à remplir ou à écrire.

Le noyau va devoir lire ce buffer, mais il ne peut pas nécessairement le faire parce qu'il ne travaille pas nécessairement dans le même espace d'adressage. Il peut y avoir un changement de la table des pages.

Pour accéder aux données, le noyau doit utiliser deux fonctions permettant de déplacer les données entre l'espace d'adressage du noyau et de l'utilisateur.

```
copy_from_user(     unsigned long dest, // addr kernel
                    unsigned long src,  // addr user
                    unsigned long len); // nb bytes

copy_to_user(       unsigned long dest,
                    unsigned long src,
                    unsigned long len);
```


Accès au device

Pour communiquer avec un device, il faut ouvrir le pseudo-fichier qui lui a été associé : fd = open("/dev/*device_name*", mode);

- fd (*file descriptor*) est un entier qui identifie l'accès dans l'application
- Par exemple pour l'opération d'écriture : write(fd, buf, count);

Dans le noyau, toutes les opérations du driver reçoivent une structure file qui a été créée par le noyau pour ce file correspondant au fd rendu à l'utilisateur.

Toutes les fonctions du driver reçoivent file :

```
static int
my_open_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "open()\n");
    return 0;
}
static ssize_t
my_write_function(struct file *file, const char *buf, size_t count, loff_t
*ppos) {
    printk(KERN_DEBUG "write()\n");
    return 0;
}
```

Structure file

Elle est définie dans `<linux/fs.h>`

La structure `file` est passée à toutes les opérations.
Les champs importants sont :

- `mode_t f_mode` indique le mode d'ouverture du fichier
- `loff_t f_pos` position actuelle de lecture ou d'écriture
- `unsigned int f_flags` les flags de fichiers (p.ex. O_RDONLY, ...)
- `struct file_operations *f_op` les opérations associées au fichier définies à l'enregistrement du device/
- `void *private_data` un pointeur vers des données privées propre au fichier ouvert. Nous allons utiliser ce champs pour y ranger un état spécifique.

Allocation dynamique pour le fichier

La structure file contient un champ void *private_date

```
static int
my_open_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "open()\n");
     file->private_data = kmalloc(SIZE, GFP_KERNEL);
    return 0;
}

static int
my_release_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "close()\n");
    kfree(file->private_data);
     return 0;
}
```

ioctl

ioctl Syscall

- Envoie une commande vers le pilote du périphérique
  - lire de donnée depuis le pilote
  - écrire des données dans le pilote
  - modifier le comportement ou configurer le périphérique

- Fonction avec un nombre d'arguments variable
  L'interprétation de la requête (cmd) et le nombre d'argument dépend du pilote et du périphérique.
  Deux implémentations : compat_ioctl est la première version avec un « big lock ».

```
int unlocked_ioctl(int fd, unsigned long cmd, ...);
int compat_ioctl(int fd, unsigned long cmd, ...);
```

appel système ioctl

```
struct file_operations fops =
{
     .owner              = THIS_MODULE,       /* pointeur sur le module courrant */
     .open               = my_open_function,
     .read               = my_read_function,
     .write              = my_write_function,
     .unlocked_ioctl     = my_ioctl_function,
     .release            = my_release_function  /* appelée par le dernier close */
};
```

IOCTL : input output control

- permet de faire des opérations qui ne peuvent pas être faites par les autres appels
- par exemple, dans le cas du LCD, positionner le curseur

appel de ioctl par l'utilisateur

Appel système côté utilisateur

err = unlocked_ioctl (fd, cmd, *arg*)

- fd file descriptor
- cmd unsigned long doit être **unique** dans le système
- *arg* argument optionnel (void *)
- err 0 ou en cas d'erreur -1 et errno (code d'erreur)

	3/2	13/14	8	8
cmd	sens	taille	type	num

- type : doit être différent pour chaque pilote
- num : numéro d'ordre de la commande
- taille : quantité de données échangées
- sens : sens des échanges de données, par rapport au programme utilisateur

Fabrication du paramètre cmd

cmd est obtenu par des macros de **sys/ioctl.h**

- _IO (type, num)
- _IOW (type, num, taille)
- _IOR (type, num, taille)
- _IOWR (type, num, taille)

Le sens du transfert est du point de vue de l'application

Pour garantir l'unicité le type doit être choisi après consultation du fichier

⇒ linux/Documentation/ioctl/ioctl-number.txt

- num est un nombre séquentiel
- taille c'est la quantité de données échangées

```
#define    TIOCSETAF  _IOW('t', 22, struct termios) /* drn out, fls in, set */
#define    TIOCGETD   _IOR('t', 26, int)    /* get line discipline */
```


Convention de nommage de la commande

DRIVER_NAME_IOCXXXXX

X Type d'opération

XXXX Nom de la commande

Si l'argument est entier

Tell: donne l'argument

Query: demande une réponse dans la valeur de retour

sHift: T + Q atomique

Si l'argument est un pointeur

Set: définir

Get: obtenir

eXchange G + S atomique

Paramètre arg

unsigned long

Absent si rien à échanger

On a toute liberté sur la signification et l'utilisation de donnée fournie au pilote

- Adresse de données fournies au pilote
- Adresse à laquelle le pilote renvoie des données
- Pointeur sur une structure
- Valeur entière

Les adresses sont dans l'espace utilisateur

Gestion de ioctl coté noyau

```
static int
lcd_ioctl( struct file *filep,
           unsigned int cmd,
           unsigned long arg)
```

La commande et l'argument sont ceux de l'utilisateur

On doit vérifier la validité de la commande

- _IOC_DIR(cmd)
- _IOC_TYPE(cmd)
- _IOC_NR(cmd)
- _IOC_SIZE(cmd)

Décodage de la commande

driver.h

```
struct cord_xy {
      int line;
      int row;
} cord_xy;
#define IOC_MAGIC           't'
#define LCDIOCT_CLEAR       _IO(IOC_MAGIC, 20)
#define LCDIOCT_SETXY       _IOW(IOC_MAGIC, 21, struct cord_xy)
```

driver.c

```
static long ioctl_lcd(struct file *file, unsigned int cmd, unsigned long arg)
{
      printk(KERN_DEBUG "Ioctl_lcd ! \n");
      struct cord_xy cord;

      if(_IOC_TYPE(cmd) != IOC_MAGIC) // Check the magic number of the device
            return -EINVAL;

      switch(cmd){
            case LCDIOCT_CLEAR:
                  file->f_pos = 0;
                  lcd_clear();
                  break;
            case LCDIOCT_SETXY:
                  if(copy_from_user(&cord, (void*)arg, _IOC_SIZE(cmd)) != 0)
                        return -EINVAL;
                  kline = cord.line;
                  krow = cord.row;
                  break;
            default: return -EINVAL;
      }
      return 0;
}
```


TME

Dans le prochain TME,
Partant d'un programme pilotant l'afficheur LCD en mode USER (root), vous devrez écrire un driver pour le contrôler comme un tty.