



UNIVERSITÉ
CÔTE D'AZUR

Pilotes et Périphériques



Présentation: Stéphane Lavirotte
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
**Olivier Dalle, Erick Galesio, Fabrice Huet, Stéphane Lavirotte,
Michael Opdenacker, Jean-Paul Rigault**

Mail: Stephane.Lavirotte@unice.fr
Web: <http://stephane.lavirotte.com/>
Université Côte d'Azur



Y'a-t-il un Pilote ?

...dans l'ordinateur



- ✓ **Sous Unix, tout est fichier**
 - Donc l'accès à un périphérique se fait à travers un fichier
- ✓ **Les fichiers spéciaux sont dans le répertoire `/dev` (convention)**
- ✓ **Entrées / Sorties sur un fichier spécial:**
 - Pas de primitive dédiée
 - Ouverture d'un fichier spécial = initialisation d'une structure dans le noyau
 - E/S sur le fichier spécial utilisent cette structure = dialogue avec le périphérique se fait par des primitives d'E/S classiques



✓ 3 types de périphériques:

– Périphériques caractère:

- Données non structurées, souvent séquentielles
- Accès caractère par caractère
- Ex: clavier, souris, port parallèle, IrDA, Bluetooth, console, ...

– Périphériques bloc:

- Données structurées de taille fixe
- E/S se font au travers du buffer cache
- Ex: disques durs, disques mémoires, périphériques de loopback (images de systèmes de fichiers)...

– Périphériques réseau

- Ne fonctionnent pas sur le même modèle (pas d'entrée dans `/dev`)
- Utilisation de la commande `ifconfig -a` pour les lister
- Ex: `ppp0`, `eth1`, `usbnet`, `irda0`



Exemple de fichiers de /dev

✓ Exemple de fichiers de /dev (`ls -l /dev`)

– Pilotes de type Caractère

```
crw-rw---- 1 root daemon 6, 0 May 5 2016 /dev/lp0
crw-rw-rw- 1 root root 1, 3 May 5 2016 /dev/null
crw--w---- 1 eg tty 4, 1 May 5 2016 /dev/tty1
crw-rw-rw- 1 root root 3, 1 May 5 2016 /dev/ttyp1
crw-rw-rw- 1 root root 1, 5 May 5 2016 /dev/zero
```

– Pilote de type Bloc

```
brw-rw---- 1 root disk 3, 0 May 5 2016 /dev/hda
brw-rw---- 1 root disk 3, 1 May 5 2016 /dev/hda1
brw-rw---- 1 root disk 3, 2 May 5 2016 /dev/hda2
brw-rw---- 1 root disk 3, 3 May 5 2016 /dev/hda3
brw-rw---- 1 root floppy 2, 0 May 5 2016 /dev/fd0
Brw-rw---- 1 root disk 7, 0 May 5 2016 /dev/loop0
brw-rw---- 1 root disk 8, 0 May 5 2016 /dev/sda
brw-rw---- 1 root disk 8, 1 May 5 2016 /dev/sda1
```



Majeur / Mineur

✓ Chaque périphérique est identifié par 2 numéros

– Majeur: c'est en fait le numéro du pilote

```
0 crw-rw---- 1 root daemon 6, 0 May 5 2016 /dev/lp0
0 brw-rw---- 1 root disk 8, 0 May 5 2016 /dev/sda
```

– Mineur: c'est une fonction ou un périphérique physique géré par le pilote

▪ Fonctions différentes du même driver

```
0 crw-rw-rw- 1 root root 1, 3 May 5 2016 /dev/null
0 crw-rw-rw- 1 root root 1, 5 May 5 2016 /dev/zero
```

▪ Périphériques différents gérés par le même driver

```
0 brw-rw---- 1 root disk 8, 0 May 5 2016 /dev/sda
0 brw-rw---- 1 root disk 8, 16 May 5 2016 /dev/sdb
```

✓ Nombre maximum de périphériques (max. numéro majeur)

– en 2.0: 128

– en 2.2 et 2.4: 255 majeurs (et 255 mineurs)

– en 2.6: 4096 (mais limite extensible)

✓ Numéro/Périphérique: Documentation/devices.txt



Définition d'un Fichier dans /dev

- ✓ **Les fichiers de périphériques**
 - Ne sont pas créés (par défaut) lorsqu'un pilote est chargé
 - Ils doivent être créés par avance
- ✓ **En C (dans le noyau, même si n'est pas au noyau de créer les entrées de /dev):**

```
#include <linux/fs.h>
int (*mknod) (struct inode *, struct dentry *, int, dev_t);
```

```
#include <linux/kdev_t.h>
#define MINORBITS      20
#define MINORMASK      ((1U << MINORBITS) - 1)
#define MAJOR(dev)      ((unsigned int)((dev) >> MINORBITS))
#define MINOR(dev)      ((unsigned int)((dev) & MINORMASK))
#define MKDEV(ma, mi)   (((ma) << MINORBITS) | (mi))
```

- ✓ **Sous shell:**

```
mknod /dev/... b <major> <minor> pour un périphérique en mode bloc
mknod /dev/... c <major> <minor> pour un périphérique en mode char
```



Mise en œuvre d'un Pilote

de périphérique...



Enregistrement d'un Pilote (Old Fashion ≤ 2.6)

- ✓ Tout d'abord il faut créer l'(les)entrée(s) correspondante(s) dans `/dev` (sous Shell, pas dans le code du driver !)
- ✓ Pour enregistrer un pilote système, il faut le déclarer¹:

```
#include <linux/fs.h>
int register_chrdev(unsigned int major,
                    const char* name,
                    const struct file_operations *fops);
```

- `major` **est le numéro majeur désiré.**
- `name` **est le nom qui permet d'identifier le pilote** (`/proc/devices`)
- `fops` **est un pointeur sur une structure contenant des pointeurs sur fonctions décrivant l'implémentation des primitives de base d'E/S ou de gestion des périphériques.**

1. De façon symétrique `register_blkdev` permet l'enregistrement d'un périphérique bloc

Trouver un Numéro Majeur Libre (Old Fashion ≤ 2.6)

- ✓ De moins en moins de numéros majeurs sont disponibles
 - Il n'est pas recommandé d'en prendre un arbitrairement, car il peut rentrer en conflit avec un autre pilote (standard ou spécifique)
- ✓ **Solution: laisser `register_chrdev` en trouver un libre dynamiquement pour vous !**
 - Mettre le paramètre `major` à 0 au moment de l'enregistrement
 - `major = register_chrdev (0, "foo", &name_fops);`
- ✓ **Problème: vous ne pouvez pas créer d'entrées `/dev` par avance !**
- ✓ **Cependant, le script chargeant le module peut se servir de `/proc/devices`:**

```
module=foo; device=foo
insmod $module.ko
major=`awk "\\$2==\"$module\" {print \\$1}" /proc/devices`
mknod /dev/foo0 c $major 0
```

Libération d'un Pilote (Old Fashion ≤ 2.6)

- ✓ **Lorsqu'un pilote n'est plus nécessaire, il peut être libéré de la mémoire par la primitive `unregister_chrdev`**

```
void unregister_chrdev(unsigned int major,  
                      const char* name);
```

- `major` **est le numéro majeur désiré.**
- `name` **est le nom qui permet d'identifier le pilote**

- ✓ **Attention:**

- **Ne pas libérer le pilote si certains périphériques sont encore en cours d'utilisation !!!**
- **Oublier de libérer un pilote (lors du déchargement d'un module) \Rightarrow reboot pour charger de nouveau le module !!!**
- **Cette méthode « Old Fashion » ne marche qu'avec des numéros majeur < 256**

Gestion Pilotes

New Fashion ≥ 2.6

✓ Fonctions

```
int register_chrdev_region(dev_t first, unsigned int count,  
    char *name);  
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,  
    unsigned int count, char *name);  
void unregister_chrdev_region(dev_t first, unsigned int  
    count);
```

✓ Exemple d'utilisation

```
if (my_major) {  
    dev = MKDEV(my_major, my_minor);  
    result = register_chrdev_region(dev, my_nr_devs, "mydriver");  
} else {  
    result = alloc_chrdev_region(&dev, my_minor, my_nr_devs,  
        "mydriver");  
    my_major = MAJOR(dev);  
}  
if (result < 0) {  
    printk(KERN_WARNING "mydriver: can't get major %d\n",  
        my_major);  
    return result;  
}
```



Au cœur d'un Pilote

Dissection d'un pilote de périphérique

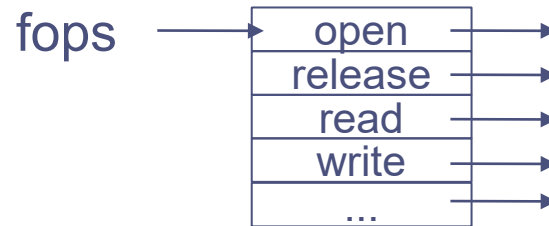


Espaces de Travail

- ✓ **Trois espaces de travail:**
 - **l'espace matériel (hardware space):** correspond aux périphériques
 - Cartes connectées à la carte mère par l'intermédiaire des ports (slots) AGP, ISA, PCI, PCI Express
 - Les appareils reliés à l'ordinateur par l'intermédiaire des différents ports tels que PS/2, USB, firewire, parallèle et série.
 - **l'espace noyau (kernel space)**
 - Espace du noyau Linux
 - Toute erreur est fatale au système
 - **l'espace utilisateur (user space)**
 - Espace des programmes utilisateurs
 - Une erreur est fatale pour le processus, mais ne remet pas en cause l'intégrité du système.
- ✓ **Un pilote de périphérique utilise des structures et des fonctions provenant du noyau.**

Opérations sur les fichiers: Généralités 1/2

- ✓ Lors de la déclaration d'un pilote, on doit passer une structure décrivant le comportement du pilote en fonction des opérations désirées sur les périphériques.



- ✓ Cette table de pointeurs est accessible depuis toutes les primitives de la table \Rightarrow les pointeurs sur fonction peuvent être changés à tout moment (en général au moment de l'open) pour changer le comportement du pilote pour un périphérique particulier.
 - /dev/null
 - /dev/zero



Opérations sur les fichiers: Généralités 2/2

✓ **La structure `file_operations` (`linux/fs.h`) est définie comme:**

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb*, const struct iovec *, unsigned longsize_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb*, const struct iovec *, unsigned longsize_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compact_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ...
};
```

✓ **Si une opération n'est pas supportée,**

- L'appel système correspondant produira l'erreur `EINVAL`.



Déclaration de `file_operations`

- ✓ **Structure qui doit être fournie dans le module**
 - **Solution 1:** On met des NULL pour les champs non définis
 - **Solution 2:** Pour être portable, utiliser l'indexation (notation pointée)
 - `.owner, .read, .write, .open, .release, ...`
 - **Exemple de déclaration de structure `file_operations`**

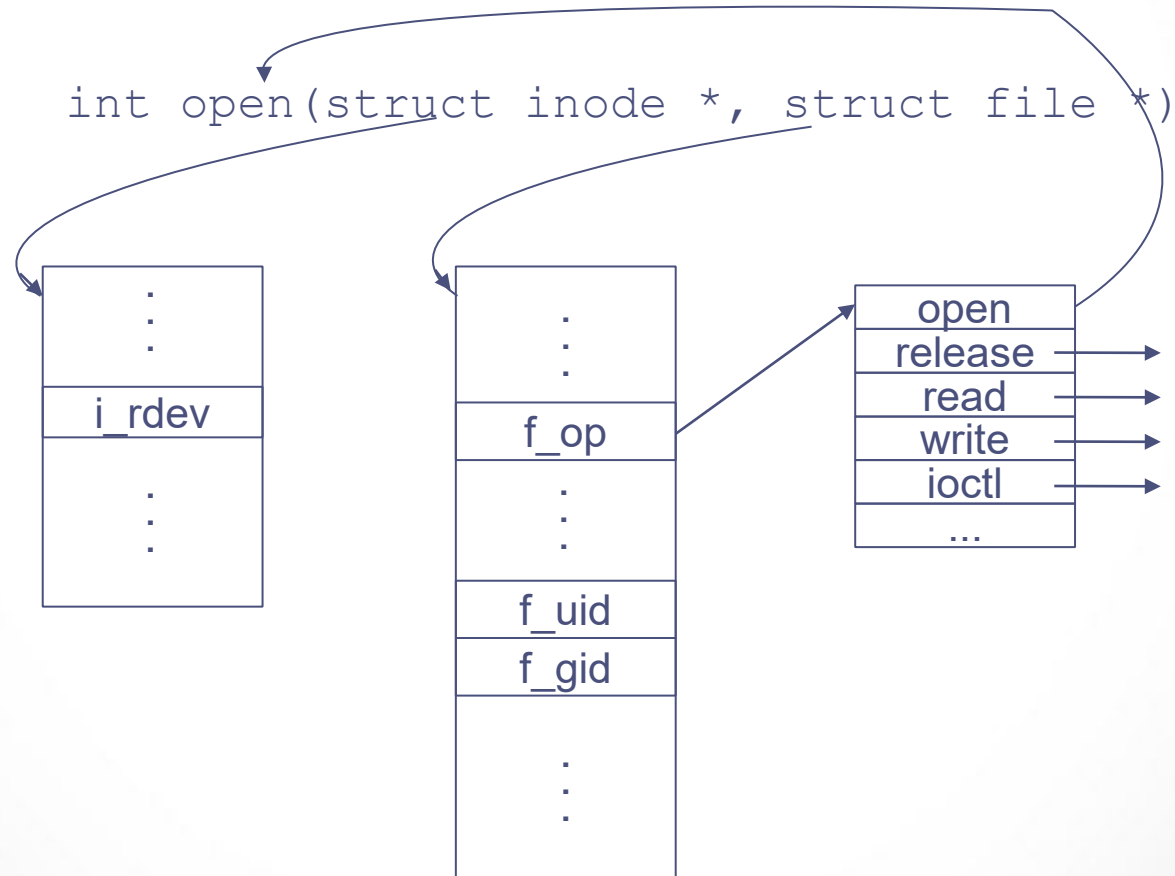
```
struct file_operations mon_fops = {  
    .owner = THIS_MODULE,  
    .read = mon_module_read,  
    .write = mon_module_write,  
    .open = mon_module_open,  
    .release = mon_module_release,  
    ...  
};
```

ou

```
struct file_operations mon_fops = {  
    owner : THIS_MODULE,  
    read : mon_module_read,  
    write : mon_module_write,  
    open : mon_module_open,  
    release : mon_module_release,  
    ...  
};
```



Appel à open





Structure File

- ✓ Définie dans `<linux/fs.h>`
- ✓ Structure créée par le système durant l'appel à `open()`
 - Représente les fichiers ouverts.
 - Les pointeurs vers cette structure sont appelés "fips"

```
struct file {  
    const struct file_operations  *f_op;  
    atomic_t                      f_count;  
    unsigned int                  f_flags;  
    mode_t                       f_mode;  
    loff_t                       f_pos;  
    void                          *private_data;  
    ...  
};
```

Opérations sur les fichiers:

Ouverture

```
int (*open)(struct inode *inode, struct file *file);
```

- Vérifie les erreurs possibles sur le périphérique (pas prêt, problème hardware)
 - Initialise le périphérique si c'est la première ouverture
 - « Jongle » éventuellement avec le champs `f_op` en fonction du n° mineur
 - Incrémente un compteur d'utilisation pour éviter la destruction du pilote si certains périphériques sont en cours d'utilisation
 - Peut éventuellement sauvegarder des informations dans le champ `private_data` présent dans la structure `file`.
- ✓ Jusqu' à 2.4 inclus, si on utilise un module chargeable utiliser la macro `MOD_INC_USE_COUNT` qui interdira le déchargement du module (et donc l'exécution de `unregister_xxxdev` qui se trouve généralement dans la procédure `cleanup_module`)

Opérations sur les fichiers:

Fermeture

```
int (*release) (struct inode *inode, struct file *file);
```

- **décrémente le compteur d'initialisation**
 - **libère les informations qui auraient pu être sauvegardées dans le champ `private_data`**
- ✓ **Jusqu'en 2.4 inclus, si on utilise un module chargeable utiliser la macro `MOD_DEC_USE_COUNT` qui décrémente le compteur du module. Si ce compteur n'est pas nul, la commande `rmmod` échoue. La valeur de ce compteur est affichée dans `/proc/modules`**

Opérations sur les fichiers:

Lecture (1/2)

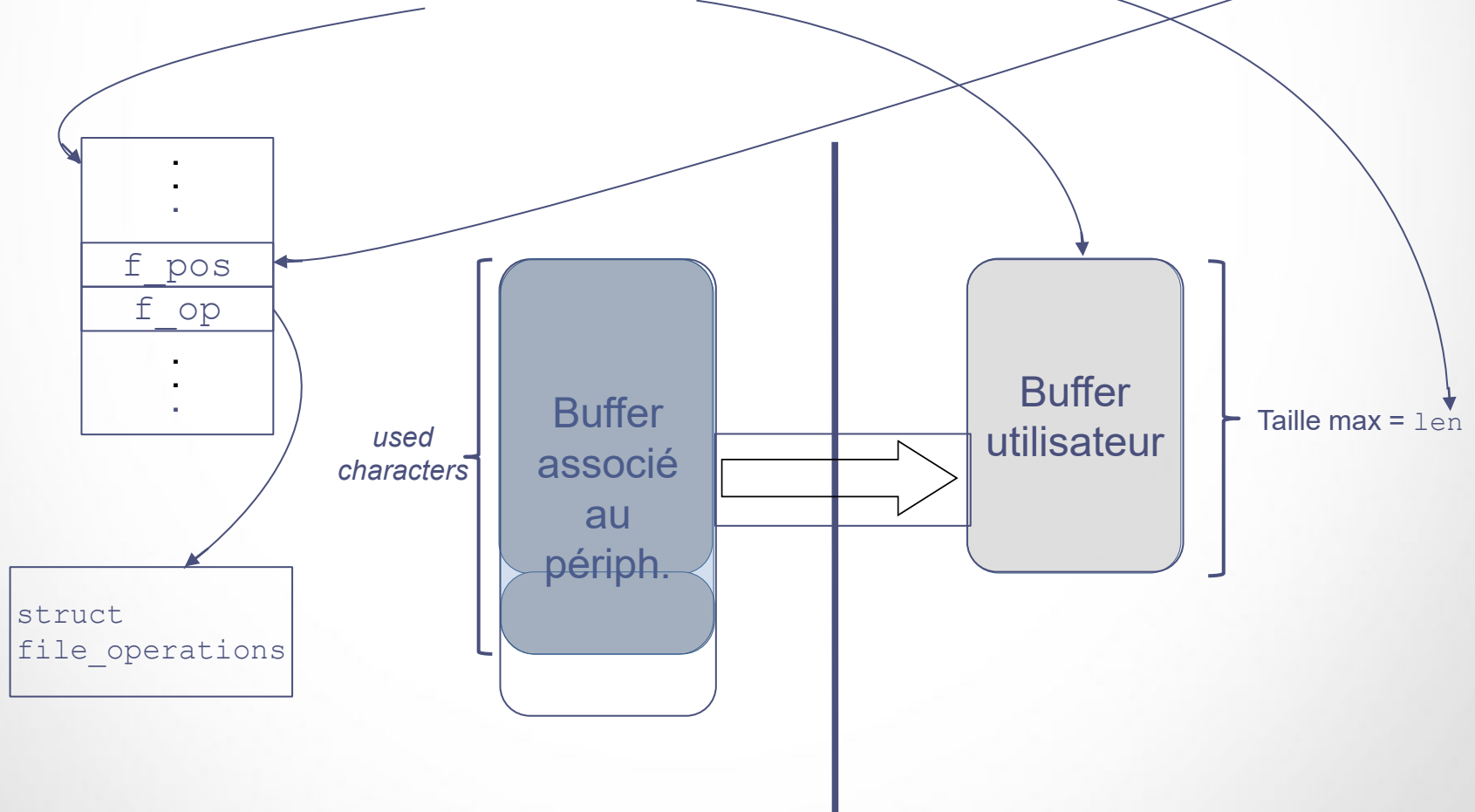
```
ssize_t (*read)(struct file *f, char __user *buff, size_t len, loff_t  
    *ptr);
```

- ✓ **La fonction `read` doit placer les caractères lus dans `buff`.**
Plusieurs cas pour la valeur de retour `n`:
 - `n = len` : **pas de problème**
 - `n < len` : **l'appel sera peut-être continué automatiquement par la primitive de lecture utilisée (e.g. `fread`)**
 - `n = 0` : **fin de fichier \Rightarrow l'appel système se termine**
 - `n < 0` : **code d'erreur**

- ✓ **Attention:** `buff` est dans l'espace utilisateur. Utiliser la fonction `put_user` pour écrire le résultat.

Opérations sur les fichiers: Lecture (2/2)

```
ssize_t read(struct file *f, char __user *buff, size_t len, loff_t *ppos)
```



Opérations sur les fichiers:

Ecriture

```
ssize_t (*write)(struct file *f, const char __user *buff, size_t  
len, loff_t *ptr);
```

✓ **La fonction** `write` **doit lire les caractères dans** `buff`.

Plusieurs cas pour la valeur de retour `n`:

- `n = len` : **pas de problème**
- `n < len` : **l'appel peut être relancé**
- `n = 0` : **pas une erreur (bloqué)**
- `n < 0` : **code d'erreur**

✓ **Attention:** `buff` **est dans l'espace utilisateur. Utiliser la fonction** `get_user` **pour chercher le contenu du buffer.**

Opération pour la configuration d'un pilote: `ioctl`

- ✓ Possibilité de paramétrage d'un pilote au lancement
 - Utilisation des paramètres de chargement
- ✓ Mais besoin d'envoyer des commandes de contrôle au pilote pendant son exécution qui ne soient ni des lectures ni des écritures
 - Ex: Changement de résolution d'une webcam
- ✓ Suppression de la méthode `ioctl` depuis 2.6.36
 - Remplacement par `unlocked_ioctl`
 - Evite le problème de Big Kernel Lock (BLK)
 - <http://lwn.net/Articles/119652/>

Résumé de la Création d'un Pilote

✓ Ecriture du Pilote

- Définir votre fonction d'initialisation du module et appeler `register_chrdev()`:
 - Donner un numéro majeur, ou 0 (automatique)
- Définir la fonction de sortie du module, et y appeler la fonction `unregister_chrdev()`
- Définir vos opérations sur fichier (`fops`)
 - Passer le `fops` à votre `register_chrdev`
 - Faire l'implémentation de chacune des fonctions pointées par `fops`

✓ Utilisation du pilote

- Créer l'entrée dans `/dev/`
- Charger votre module
- Il ne reste plus qu'à utiliser votre pilote !!

Pilote en Mode Noyau vs Pilote en Mode Utilisateur

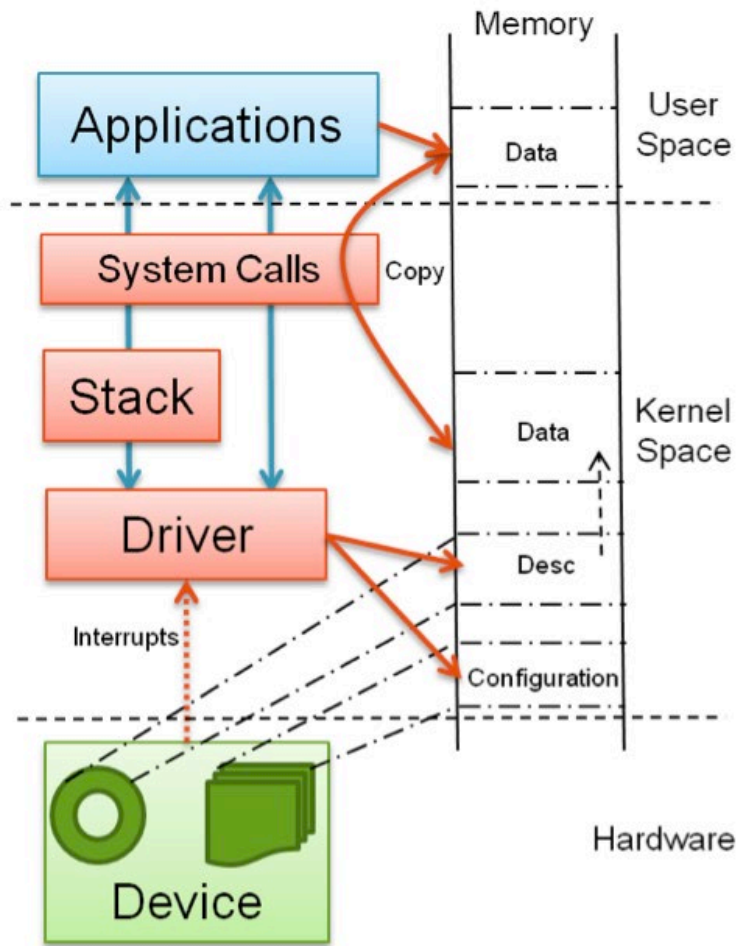


Figure 1: Kernel space network driver

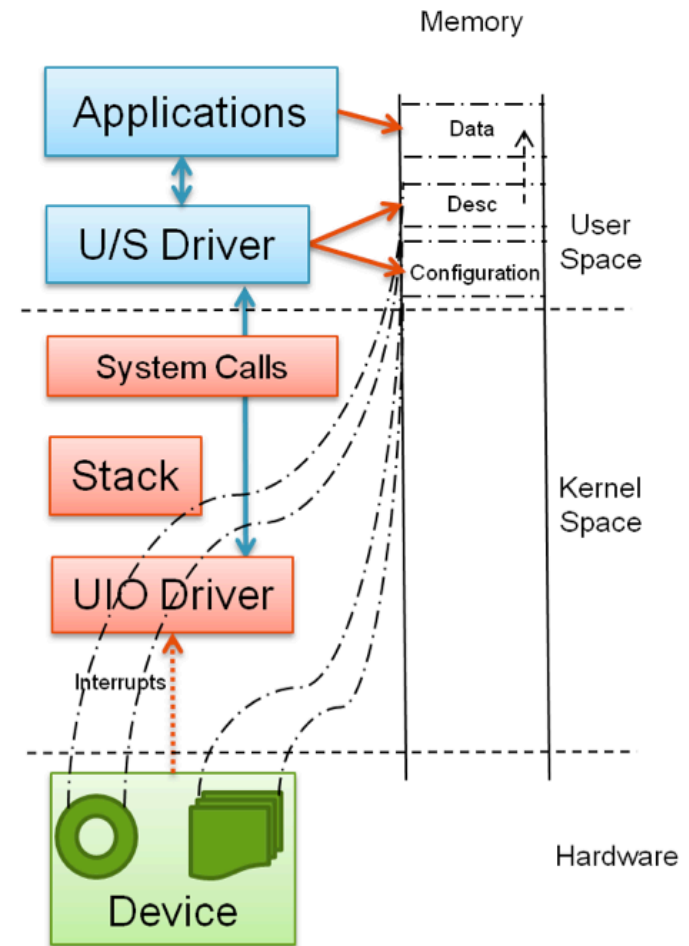


Figure 2: User space network driver



Pilote en Mode Utilisateur

Avantages

Le développement de pilotes se fait généralement en mode noyau, mais on peut aussi construire un pilote de périphérique en mode utilisateur

✓ Avantages du mode utilisateur:

- Accès complet à la bibliothèque C !!!
- Possibilité de manipuler des nombres flottants
- Si le pilote se plante, il suffit de le tuer (sinon c'est la machine qui plante)
- Possibilité de correction avec les outils standards de *debug*. Le *debug* du noyau est par contre compliqué
- Un pilote en mode utilisateur peut être *swappé* s'il ne sert pas

Pilote en Mode Utilisateur Inconvénients

- ✓ Par contre un pilote en mode utilisateur pose un certain nombre de problèmes:
 - Les interruptions ne sont pas accessibles
 - Les ports d'entrée/sortie ne sont accessibles qu'au travers d'appels systèmes réservés au super-utilisateur (`ioperm` et/ou `iopl`)
 - L'accès direct à la mémoire n'est possible qu'au travers de `/dev/mem` (par exemple en `mmapant`), mais ce fichier n'est accessible qu'au super utilisateur
 - Les temps de réponse sont plus longs car les transferts de données nécessitent un changement de contexte (voire même de faire revenir le pilote de *swap* !!)



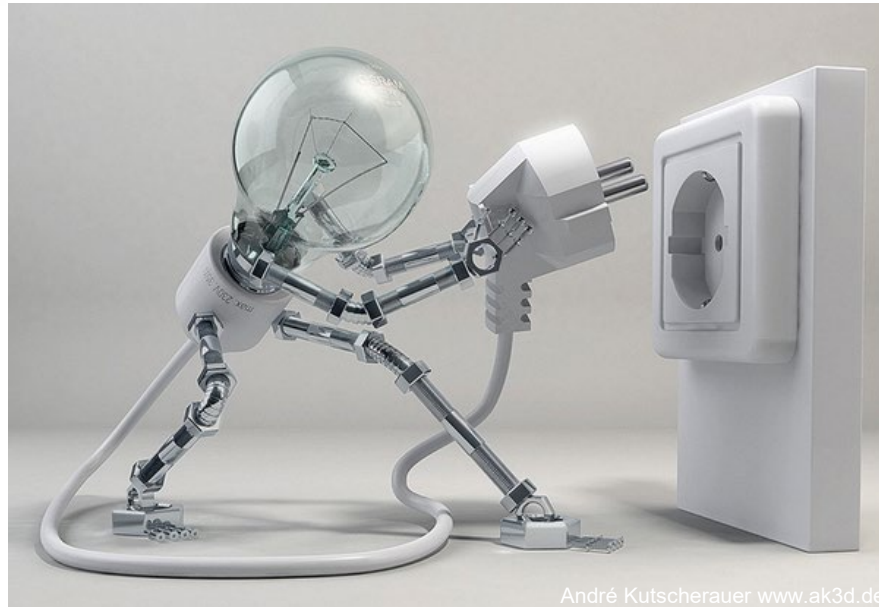
Les Pilotes et Périphériques côté « Espace Utilisateur »

Gestion des pilotes et périphériques
dans l'espace utilisateur



Gestion des Périphériques

- ✓ **Plan de la partie suivante de l'exposé:**
- ✓ **Une architecture verticale pour la gestion des périphériques**
 - Implémentation du pilote dans le noyau (section précédente)
 - Consultation et Modification des pilotes (et de l'état du système) via `/sys`
 - Gestion du branchement à chaud (`hotplug`)
 - Gestion dynamique de `/dev`



André Kutscherauer www.ak3d.de

Branchement à Chaud

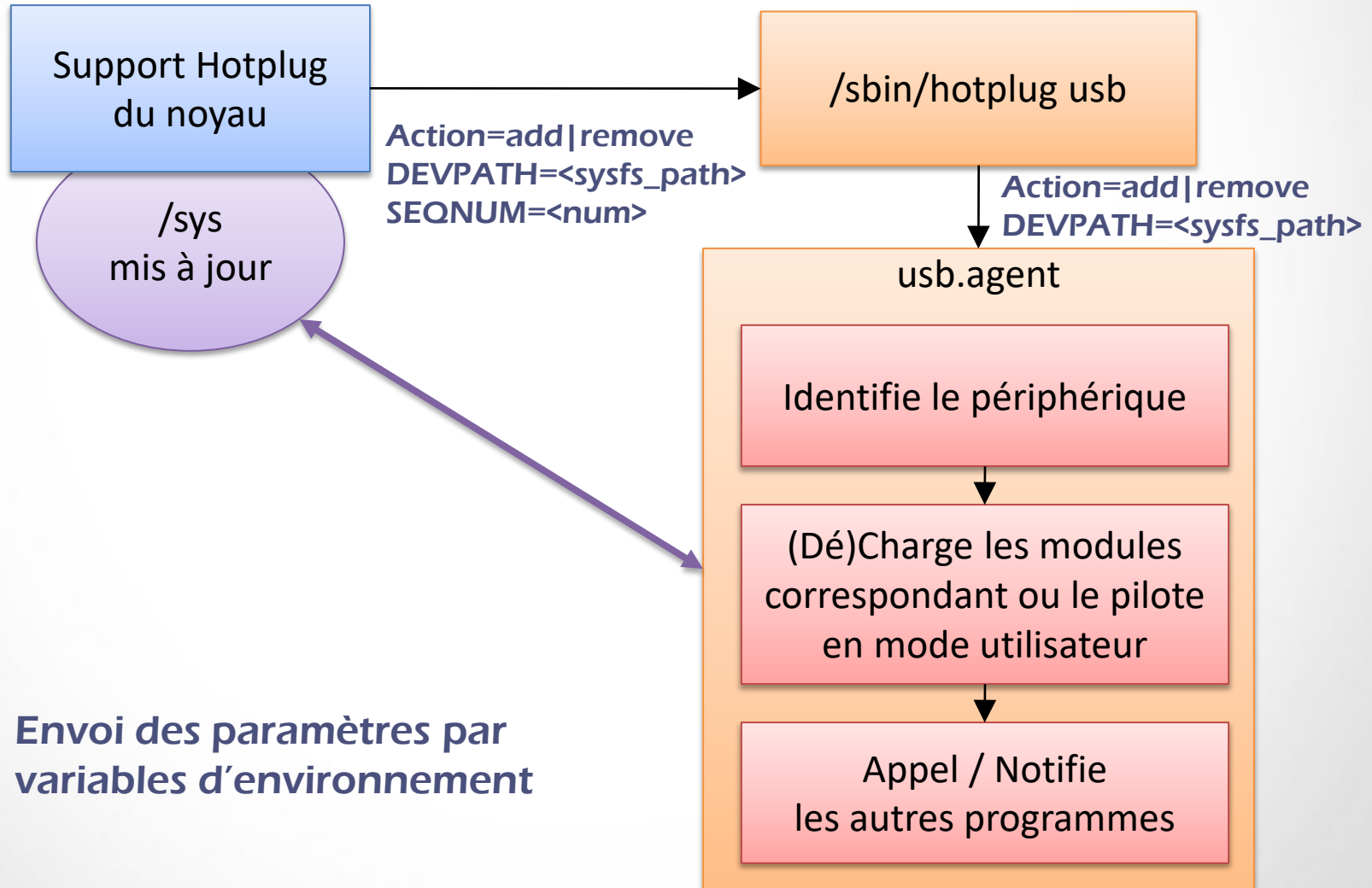
Gestion des Périphériques



Aperçu du Hotplug

- ✓ Hotplug
 - Introduit dans le noyau Linux 2.4.
 - USB a été le pionnier
 - Mécanismes noyau pour notifier les programmes de l'espace utilisateur qu'un périphérique a été inséré ou enlevé
 - Des scripts dans l'espace utilisateur prennent ensuite soin d'identifier le matériel et d'insérer/enlever les modules requis
 - Linux 2.6:
 - l'identification des périphériques est simplifiée grâce à `sysfs`
 - Rend possible le chargement de micro-logiciel (firmware)
 - Permet d'avoir des pilotes en mode utilisateur (par exemple `libsane`).
- ✓ Configuration dans le noyau:
 - `CONFIG_HOTPLUG=y` (section "General setup")
- ✓ <http://linux-hotplug.sourceforge.net/>

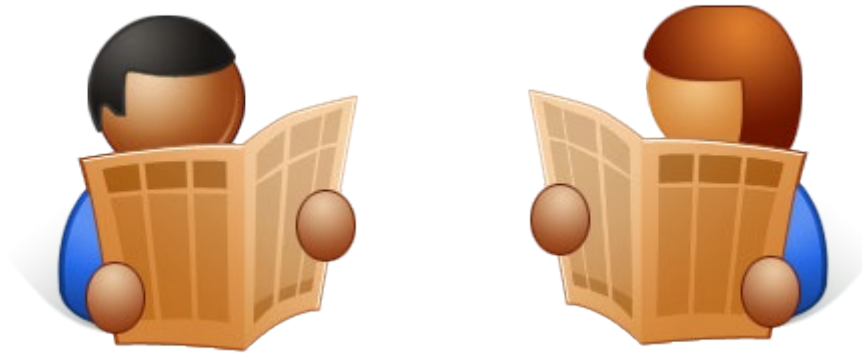
Schéma du Fonctionnement de Hotplug





Fichiers Nécessaires à Hotplug

- ✓ `/lib/modules/*/modules.*map`
 - **sortie de depmod**
- ✓ `/proc/sys/kernel/hotplug`
 - **spécifie le chemin du programme hotplug**
- ✓ `/sbin/hotplug`
 - **programme hotplug (par défaut)**
- ✓ `/etc/hotplug/*`
 - **fichiers hotplug**
- ✓ `/etc/hotplug/NAME*`
 - **fichiers spécifiques aux sous-systèmes, pour les agents**
- ✓ `/etc/hotplug/NAME/DRIVER`
 - **scripts de configuration des pilotes, invoqués par les agents**
- ✓ `/etc/hotplug/usb/DRIVER.usermap`
 - **données pour depmod pour les pilotes en mode utilisateur**
- ✓ `/etc/hotplug/NAME.agent`
 - **Agents spécifiques à des sous systèmes de hotplug.**



Gestion des Fichiers Pilotes dans l'espace Utilisateur

/dev, devfs, udev



Problèmes liés à /dev

✓ /dev

- **Beaucoup de fichiers inutiles**
 - Sur RedHat 9, /dev contenait 18.000 fichiers
 - Toutes les entrées possibles doivent être créées (à l'installation)
- **Nécessité d'avoir une autorité qui affecte les numéros majeurs**
 - Linux Assigned Names and Numbers Authority
 - <http://lanana.org/>
- **L'espace utilisateur ne sait pas quels périphériques sont présents dans le système**
- **L'espace utilisateur ne sait pas associer une entrée dans /dev avec le périphérique auquel elle se rapporte**
- **Ne peut pas être sur une partition séparée car besoin de /dev pour monter une partition**

Une Solution Transitoire: devfs ... mais des Limitations

✓ devfs

- **Device File System (Système de Fichiers de Périphériques)**
- **Utilisation de sous-répertoires pour améliorer la lisibilité**
- **Montre seulement les périphériques présents**
- **Nouvelle dénomination, ce qui pose problème dans les scripts**
 - **lien sur les anciens noms grâce au démon `devfsd`**
- **Aucune flexibilité dans le nom des périphériques (par rapport à `/dev/`)**
 - **le 1er disque IDE s'appelle soit `/dev/hda`, soit `/dev/ide/hd/c0b0t0u0`**
- **Ne permet pas l'allocation dynamique des nombres majeur et mineur.**
- **Requiert de stocker la politique de nommage des périphériques dans la mémoire du noyau.**



Une solution: udev

✓ udev

- **Vise à remplacer** `devfs`
- **Implémenté entièrement dans l'espace utilisateur**
- **Système de peuplement dynamique du répertoire** `/dev`
 - **S'appuie sur**
 - `hotplug` (mécanisme de détection des périphériques pouvant être connectés à chaud) et sur
 - `sysfs` (utilise les numéros majeurs et mineurs)
 - **Ajouts, suppressions et attributions de noms de périphériques**
- **Souplesse quant au nommage des périphériques**
 - **Système de règle pour le matching et le nommage**
- **Garantissant son déterminisme**
 - **Donne un nom précis à un périphérique défini**
 - Quelque soit le port de connexion
 - Quelque soit l'ordre dans lequel les périphériques sont détectés
- **Existence d'une interface de programmation (API)**



Configuration de udev

✓ Fichiers de configuration

- `/etc/udev/udev.conf`
 - **Fichier de configuration.** `udevcontrol` peut modifier les propriétés en cours d'exécution de `udev`
- `/etc/udev/rules.d/`
 - **Spécification des règles de création des fichiers de périphérique**
- `/etc/udev/permissions.d`
 - **Fichier définissant les permissions des fichiers de périphérique**

✓ Répertoire des fichiers de périphériques

- Par défaut `/udev`, mais dépendant des distributions
- Sous Debian, directement accessible sous `/dev`



Les outils pour udev

✓ **udevstart**

- Rempli le répertoire initial des périphériques avec des entrées valides trouvées dans l'arbre de périphériques de `sysfs`.

✓ **udevinfo**

- Obtenir les informations sur l'arborescence et les valeur des variables pour un périphérique donné
- `udevinfo -a -p $(udevinfo -q path -n /dev/video0)`

✓ **udevtest**

- Permet de voir ce que vos règles `udev` génèrent effectivement
- `udevtest /class/video4linux/video0`



✓ Construction d'une règle

- **Attention matching des règles du périphérique ou de son parent**

```
udevinfo -a -p $(udevinfo -q path -n /dev/lp0)
```

```
looking at device '/class/usb/lp0':
```

```
    KERNEL=="lp0"
```

```
    SUBSYSTEM=="usb"
```

```
    DRIVER==""
```

```
    ATTR{dev}=="180:0"
```

```
looking at parent device
```

```
    '/devices/pci0000:00/0000:00:1d.0/usb1/1-1':
```

```
    SUBSYSTEMS=="usb"
```

```
    ATTRS{manufacturer}=="EPSON"
```

```
    ATTRS{product}=="USB Printer"
```

```
    ATTRS{serial}=="L72010011070626380"
```

✓ Documentation

- http://www.reactivated.net/writing_udev_rules.html



Exemples d'Utilisation de `udev`

- ✓ **Connexion de périphériques à chaud (usb)**
 - Détection dépendant de l'ordre de connexion
 - Volonté de donner toujours le même nom à un périphérique
 - ...

- ✓ **Deux cartes d'un même type dans une machine**
 - Ordre de détection qui peut être aléatoire (suivant les conditions de boot)
 - Ex: deux cartes tuner TV (satellite, TNT)

Exemple de Règles udev

Disque Amovible

✓ Informations à la détection du disque amovible

– Clé USB, disque dur dans un boîtier USB, ...

```
usb 4-6.3: new high speed USB device using ehci_hcd and address 4
usb 4-6.3: configuration #1 chosen from 1 choice
usb 4-6.3: New USB device found, idVendor=067b, idProduct=3507
usb 4-6.3: New USB device strings: Mfr=1, Product=2,
    SerialNumber=0
usb 4-6.3: Product: Mass Storage Device
usb 4-6.3: Manufacturer: Prolific Technology Inc.
```

✓ Règles pour la détection d'un disque amovible

– 99-usbdisk.rules **dans** /etc/udev/rules.d/

```
SUBSYSTEM=="block", SYSFS{idVendor}=="067b", \
    SYSFS{idProduct}=="3507", NAME="%k", \
    SYMLINK+="icybox", RUN+="/bin/mount -t xfs \
    /dev/icybox /mnt/mydd"
```

Exemple de Règles udev

Cartes DVB

- ✓ Règles pour assurer l'ordre de détection des deux cartes dvb
 - Règle pour la carte AverMedia DVB-T

```
SUBSYSTEM=="dvb", ATTRS{vendor}=="0x1131", ATTRS{device}=="0x7133", \
PROGRAM="/bin/sh -c 'K=%k; K=${K#dvb}; printf dvb/adapter0/%%s \
${K#*.} '", NAME="%c", OPTIONS+="last_rules"
```

```
KERNEL=="video*", SUBSYSTEM=="video4linux", ATTR{name}=="saa7133*", \
SYMLINK+="tnt"
```

```
KERNELS=="input*", ATTRS{name}=="saa7134*", NAME="input/event3", \
SYMLINK+="input/ir_dvb-t"
```

- Règle pour la carte Hauppauge DVB-S

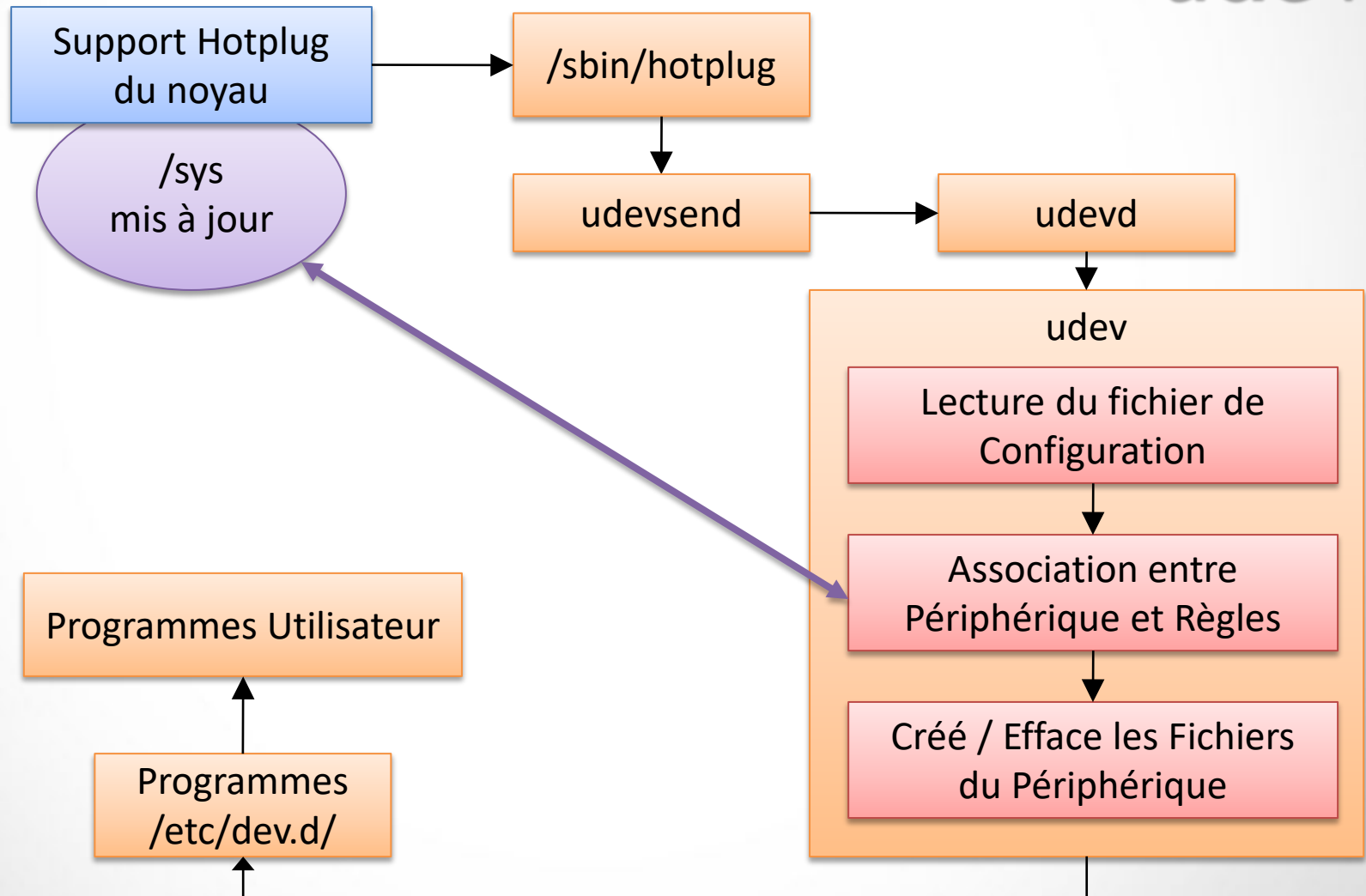
```
SUBSYSTEM=="dvb", ATTRS{vendor}=="0x1131", ATTRS{device}=="0x7146", \
PROGRAM="bin/sh -c 'K=%k; K=${K#dvb}; printf dvb/adapter1/%%s \
${K#*.} '", NAME="%c", OPTIONS+="last_rules"
```

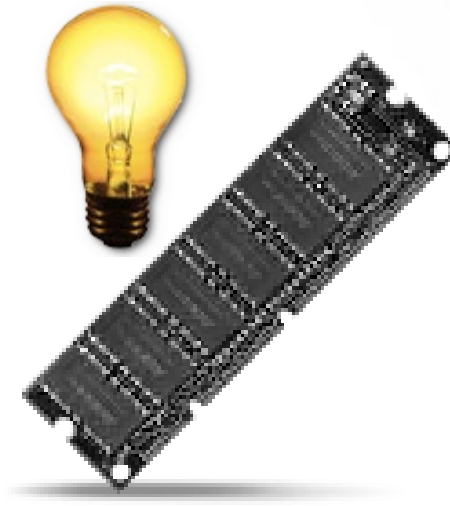
```
KERNEL=="video*", SUBSYSTEM=="video4linux", ATTR{name}=="av7110", \
SYMLINK+="sat"
```

```
KERNELS=="input*", ATTRS{name}=="DVB*", SYMLINK+="input/ir_dvb-s"
```



Schéma du fonctionnement de udev





Compléments pour le TD

Allocation dynamique de mémoire



Allocation Dynamique

✓ Malloc

- Une fonction de la bibliothèque C, non disponible dans le noyau (comme pour `printf`)

✓ Une solution: `kmalloc`

- utilisée pour l'allocation dynamique de structures dans le noyau
- ressemble à l'allocation dynamique de structures dans l'espace utilisateur
- Utilisation:

```
include <linux/slab.h>
void *kmalloc(size_t size, int priority)
```

- size : **taille en octet de l'espace mémoire à allouer**
- priority :
 - `GFP_KERNEL` : **allocation mémoire « normale du noyau**
 - `GFP_USER` : **allocation mémoire pour l'espace utilisateur**
 - `GFP_ATOMIC` : **allocation provenant du gestionnaire**
- **retourne un pointeur de la mémoire allouée ou NULL si non alloué.**

Libération Dynamique de Mémoire

✓ Free

- Pas plus de free que de malloc

✓ Utilisation de kfree


- utilisée pour la désallocation dynamique de structures dans le noyau
- ressemble à désallocation dynamique de structures dans l'espace utilisateur
- **Utilisation:**

```
include <linux/slab.h>  
void kfree(const void *ptr)
```

- ptr: **pointeur retourné par la fonction kmalloc()**

Copie UserSpace / KernelSpace

✓ Rappel du cours n°1 (Introduction)

- ✓ `int copy_to_user(unsigned long to, unsigned long from, unsigned long size);`
 - pour copier une zone mémoire depuis l'espace d'adressage du noyau vers l'espace d'adressage du processus
 - valeur de retour = nombre d'octets non transférés
- ✓ `int copy_from_user(unsigned long to, unsigned long from, unsigned long size);`
 - pour copier une zone mémoire depuis l'espace d'adressage du processus vers l'espace d'adressage du noyau
 - mêmes conventions
- ✓ Attention à la gestion du « buffer circulaire »
- ✓ Et réfléchissez, faites des dessins, faites marcher vos neurones svp !!! 



Primitives d'Accès Bas Niveau

- ✓ **E/S sur un périphérique sont réalisées par les primitives d'E/S standard** (`open`, `read`, `write`, ...)
- ✓ **Les paramètres d'un périphérique peuvent être modifiés**
 - **Par la primitive `ioctl`.**

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...)
```
 - **Pour contrôler le flux de données (configuration)**
 - **Remarques:**
 - Parfois le troisième paramètre est omis
 - La sémantique de `ioctl` dépend du périphérique adressé
 - \Rightarrow n'est pas POSIX



Valeurs de retour de Fonctions

- ✓ **Les valeurs suivantes sont utilisées dans certaines fonctions du noyau, mais aussi des fonctions que vous devez implémenter (liste non exhaustive) :**
 - **EPERM : droits insuffisants**
 - **EINTR : l'appel système a été interrompu par un signal avant d'avoir pu lire ou écrire quoi que ce soit**
 - **ENOMEM : mémoire insuffisante pour le noyau**
 - **EFAULT : pointeur pointant en dehors de l'espace d'adressage accessible**
 - **EBUSY: périphérique ou ressource occupé**
 - **EINVAL : argument invalide**
 - **ENOTTY : erreur sur le descripteur de fichier**
 - **EMSGSIZE: Message trop long**