Pilotes matériels sous Linux : *mode noyau* versus *mode utilisateur*

Stelian Pop <stelian.pop@fr.alcove.com>

Système d'exploitation

Interface entre :

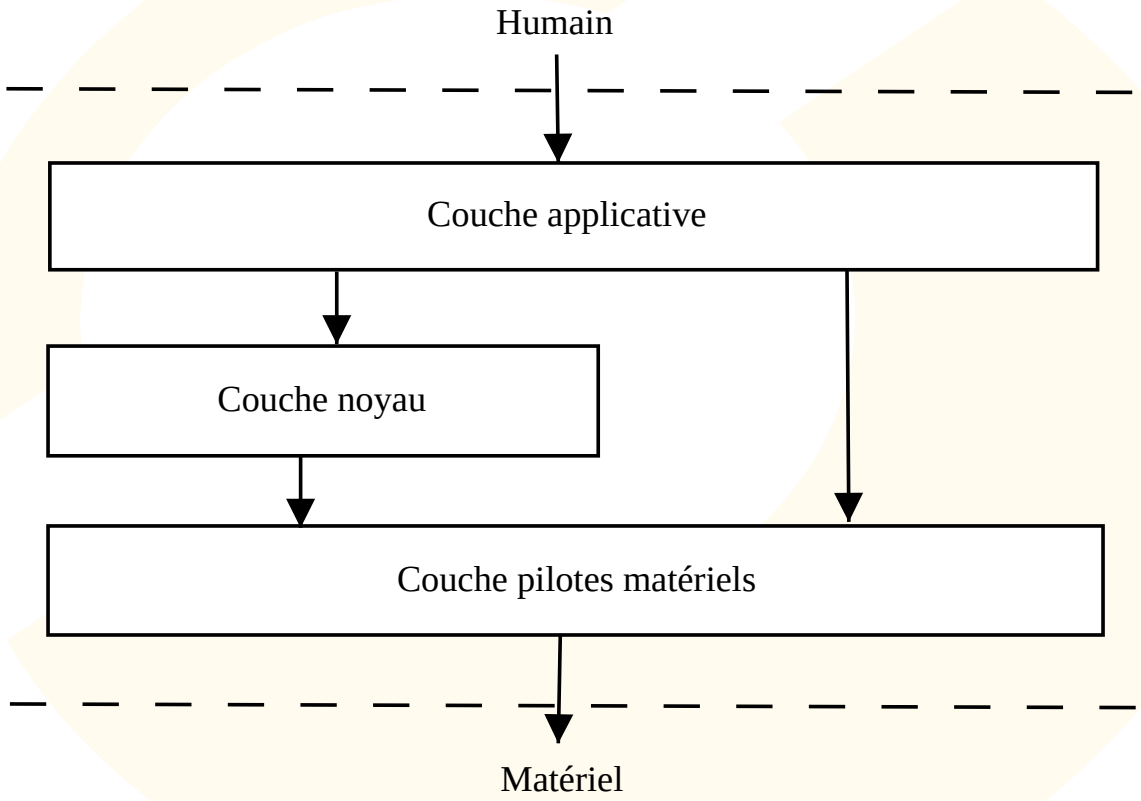
- ✓ ressources matérielles (écran, clavier, carte son...)
- ✓ utilisateur (homo sapiens)

Composé de couches logicielles :

- ✓ couche applicative (navigateur web, traitement de texte...)
- ✓ noyau (exécution des tâches, gestion de la mémoire, implémentation du réseau, accès unifié aux systèmes de fichiers...)
- ✓ pilotes matériels

Types de pilotes matériels :

- ✓ pilotes noyau (s'exécutant dans l'espace noyau)
- ✓ pilotes utilisateur (s'exécutant dans l'espace utilisateur)


Humain
Couche applicative
Couche noyau
Couche pilotes matériels
Matériel

Pilotes matériels sous Linux : *mode noyau* versus *mode utilisateur*

une comparaison pas à pas




Retour
Fermer

Exemples

Pilotes en mode noyau :

✓ répertoire drivers du noyau Linux : http://www.kernel.org

Pilotes en mode utilisateur :

✓ XFree86 : http://www.xfree86.org
✓ cdrecord : http://www.cdrecord.org
✓ SANE : http://www.mostang.com/sane
✓ sous-système USB du noyau Linux (accepte des requêtes de pilotes prototypés en mode utilisateur) : http://www.linux-usb.org

Interface avec les applications

Trois types d'API :

✓ API interne au noyau (définie par un sous-système du noyau : pilote carte son, carte réseau,...)

✓ API 'noeud de périphérique' (*device node*) (`read/write/mmap/ioctl/`... sur `/dev/foo`)

✓ API librairie utilisateur (`/usr/lib/libfoo.a`)

☛ catégories non exclusives

☛ le choix d'API influence l'implémentation

Langage de programmation

Pilotes en mode noyau :

✓ C, un peu d'assembleur

Pilotes en mode utilisateur :

✓ C, C++

✓ Java, Python

✓ Perl,

✓ ...

Bibliothèques de routines

Pilotes en mode noyau :

✓ nombre limité : traitement de chaînes, passage de paramètres...

✓ manquant : accès aux fichiers, fonctions mathématiques...

Pilotes en mode utilisateur :

✓ aucune limite...

Déboggage

Pilotes en mode noyau :

- ✓ bug = corruption, reboot nécessaire
- ✓ uniquement pile d'appels
- ✓ déboggueurs additionnels :

  kgdb http://kgdb.sourceforge.net
  - – déboggueur source (extension de gdb)
  - – deux machines reliées par câble série

  kdb http://oss.sgi.com/projects/kdb
  - – pas de source (assembleur)
  - – une seule machine

Pilotes en mode utilisateur :

- ✓ déboggage "facile" grâce aux outils existants

Accès au périphérique

☛ Protection importante pour des raisons de sécurité

Pilotes en mode noyau :

✓ accès complet aux ressources en mode noyau
✓ contrôle par :
  – droits sur le noeud de périphérique
  – système de capabilités
  – restriction à `root`
✓ point central permettant de gérer facilement les accès concurrents

Pilotes en mode utilisateur :

✓ accès au matériel permis uniquement à `root`
✓ `suid root` = risques de sécurité
✓ difficile de gérer la concurrence (sémaphores, mémoire partagée...)




Accès aux ports d'entrée/sortie

Pilotes en mode noyau :

- ✓ `request_region`, `release_region`
- ✓ `inb`, `inw`, `inl`
- ✓ `outb`, `outw`, `outl`

Pilotes en mode utilisateur :

- ✓ `iopl` (nécessite droits `root`)
- ✓ `inb`, `inw`, `inl`
- ✓ `outb`, `outw`, `outl`




Accès aux registres de configuration PCI

☛ Zone mémoire sur la carte PCI servant à configurer le périphérique, le mettre en veille...

Pilotes en mode noyau :

- ✓ `pci_enable_device`
- ✓ `pci_resource_start`
- ✓ `pci_read_config_byte` `pci_write_config_byte`
- ✓ ...

Pilotes en mode utilisateur :

- ✓ pciutils : http://atrey.karlin.mff.cuni.cz/~mj/pciutils.html
- ✓ `pci_fill_info`
- ✓ `pci_read_byte`, `pci_write_byte`
- ✓ ...

Accès à la mémoire du périphérique

☛ Pour transférer des quantités importantes de données

☛ Association à la mémoire du périphérique d'une adresse virtuelle adressable par le pilote : remappage.

Pilotes en mode noyau :

✓ `ioremap`

Pilotes en mode utilisateur :

✓ `mmap` sur `/dev/mem`

DMA

☛ Transfert de données entre un périphérique et la mémoire centrale sans l'aide du processeur

☛ Difficulté : allouer de la mémoire **physique contiguë**

Pilotes en mode noyau :

✓ `kmalloc`
✓ `get_free_pages`
✓ `virt_to_bus`

Pilotes en mode utilisateur :

✓ `memalign`
✓ `mlock`
✓ recherche dans `/dev/mem`
✓ risque de corruption mémoire...

Interruptions

☛ Moyen de communication rapide entre le périphérique et le système

Pilotes en mode noyau :

✓ `request_irq`

✓ handler d'interruption

Pilotes en mode utilisateur :

✓ impossible

Retour

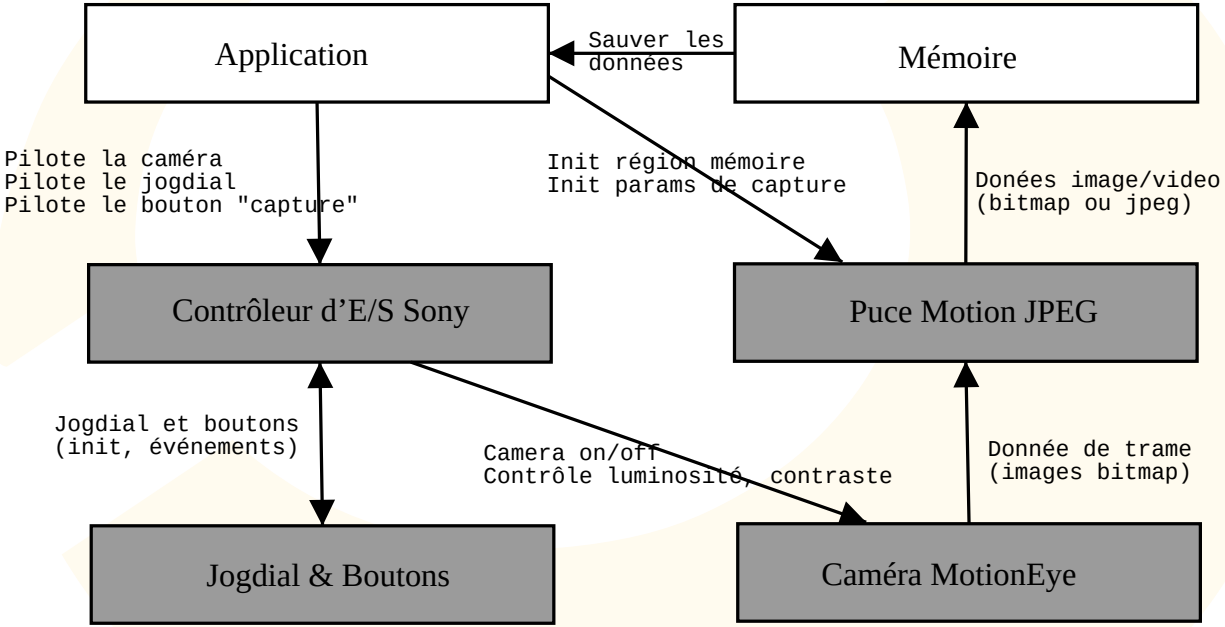
Fermer

Pilotes matériels sous Linux :
mode noyau
versus
mode utilisateur

un exemple réel :
le pilote de la caméra MotionEye

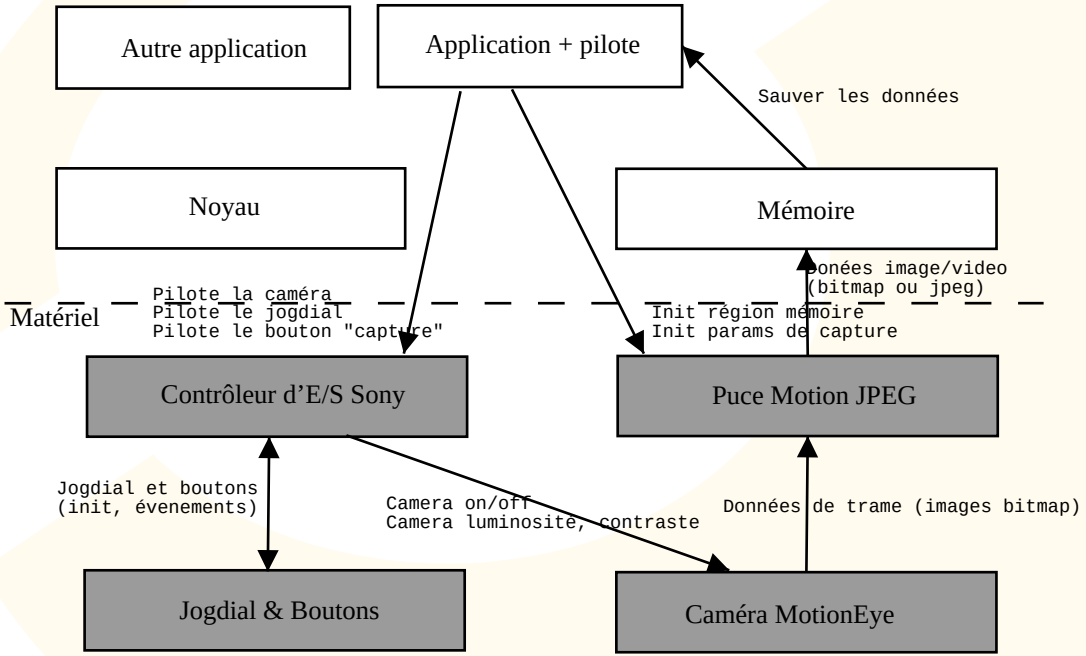
Présentation du matériel

- ✓ Sony Vaio Picturebook C1VE
- ✓ microprocesseur Crusoe par Transmeta
- ✓ ultra-portable
- ✓ lecteur MemoryStick
- ✓ molette jogdial
- ✓ webcam MotionEye


Application
Sauver les données
Mémoire
Pilote la caméra
Pilote le jogdial
Pilote le bouton "capture"
Init région mémoire
Init params de capture
Donées image/video
(bitmap ou jpeg)
Contrôleur d'E/S Sony
Puce Motion JPEG
Jogdial et boutons
(init, événements)
Camera on/off
Contrôle luminosité, contraste
Donnée de trame
(images bitmap)
Jogdial & Boutons
Caméra MotionEye

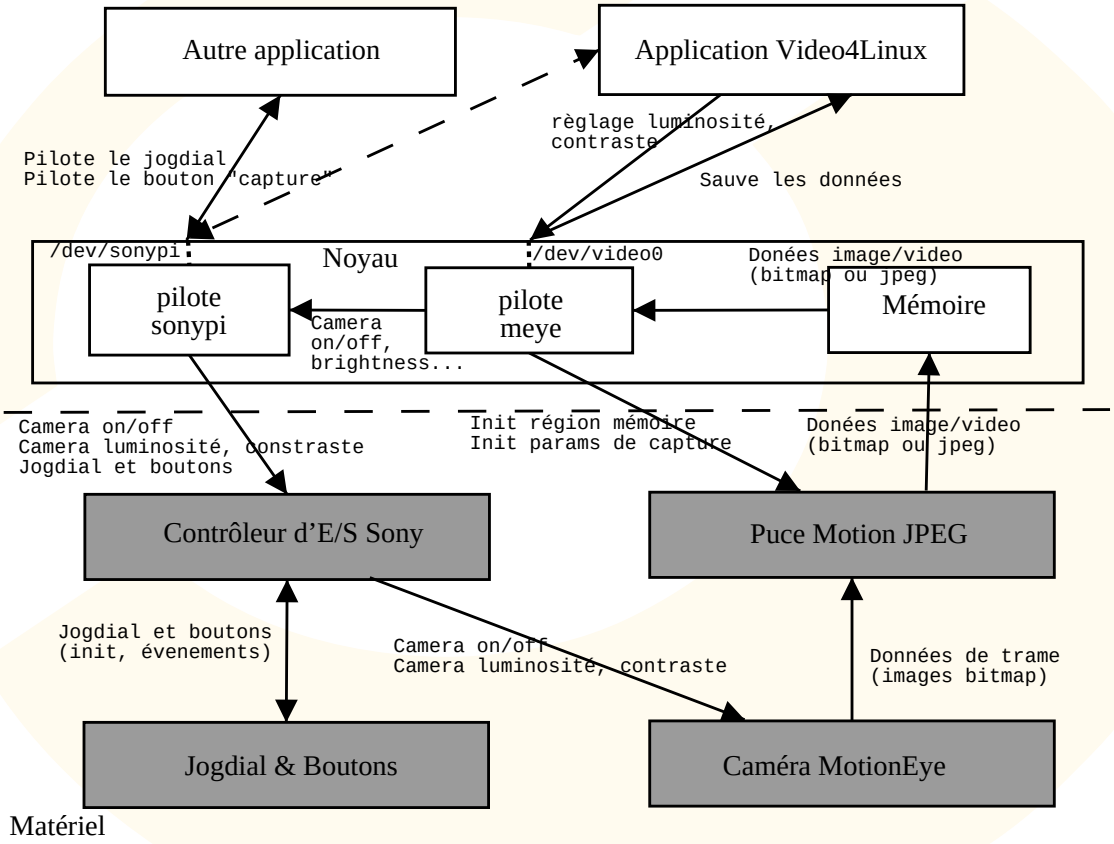
Pilote en mode utilisateur

- ✓ **capture**, par Andrew Tridgell, http://samba.org/picturebook
- ✓ monolithique (pilote + application)
- ✓ fonctionnalités :
  - afficher le flux vidéo
  - capturer le flux vidéo (non compressé ou MJPEG)
  - capturer une image (non compressée ou JPEG)
  - pilotable par le jogdial et/ou bouton ”capture”
- ✓ défauts :
  - prend le contrôle exclusif des périphériques
  - pas utilisable par d’autres applications vidéo
  - possibilité de corruption système (DMA)


Autre application
Application + pilote
Sauver les données
Noyau
Mémoire
Données image/video
(bitmap ou jpeg)
Matériel
Pilote la caméra
Pilote le jogdial
Pilote le bouton "capture"
Init région mémoire
Init params de capture
Contrôleur d'E/S Sony
Puce Motion JPEG
Jogdial et boutons
(init, évenements)
Camera on/off
Camera luminosité, contraste
Données de trame (images bitmap)
Jogdial & Boutons
Caméra MotionEye

Pilote en mode noyau

- ✓ **sonypi** & **meye**, http://www.kernel.org
- ✓ outils additionnels à http://www.alcove-labs.org
- ✓ séparation des fonctionnalités en deux pilotes
- ✓ pilote **sonypi** :
  - accessible par le noeud **/dev/sonypi**
  - donne les événements du jogdial et des boutons
- ✓ pilote **meye** :
  - implémente l'API **video4linux**
  - utilisable avec n'importe quelle application
  - accède au pilote **sonypi**
  - API supplémentaire pour le MJPEG


Autre application
Application Video4Linux
Pilote le jogdial
Pilote le bouton "capture"
règlage luminosité, contraste
Sauve les données
/dev/sonypi
Noyau
/dev/video0
Donées image/video (bitmap ou jpeg)
pilote sonypi
pilote meye
Mémoire
Camera on/off, brightness...
Camera on/off
Camera luminosité, constraste
Jogdial et boutons
Init région mémoire
Init params de capture
Donées image/video (bitmap ou jpeg)
Contrôleur d'E/S Sony
Puce Motion JPEG
Jogdial et boutons (init, évenements)
Camera on/off
Camera luminosité, contraste
Données de trame (images bitmap)
Jogdial & Boutons
Caméra MotionEye
Matériel

Changements dans l'interface

☛ Transformation depuis une application en un pilote

Fonctionnalités enlevées dans le pilote en mode noyau :

✓ affichage des données vidéo sur l'écran
✓ conversion des données vidéo YUV en données vidéo RGB
✓ sauvegarde des images et vidéo sur disque
✓ mapping des événements jogdial/boutons en actions applicatives

Changements dans l'accès aux registres PCI

Pilote en mode utilisateur :

```
int fd, bus;
DIR *dir;
u16 ven, dev;
struct dirent *dent;
char path[PATH_MAX];
...
for (bus = 0; bus < MAX_BUS; bus++) {
  sprintf(path,"/proc/bus/pci/%02d", bus);
  dir = opendir(path);
  if (!dir) continue;
  while ((dent = readdir(dir))) {
    sprintf(path,"/proc/bus/pci/%02d/%s",
            bus, dent->d_name);
    fd = open(path, O_RDWR|O_SYNC);
    if (fd == -1) continue;
    if (pread(fd, &ven, 2, PCI_VENDOR_ID) == 2 &&
        ven == PCI_VENDOR_ID_KAWASAKI &&
        pread(fd, &dev, 2, PCI_DEVICE_ID) == 2 &&
        dev == PCI_DEVICE_ID_MCHIP_KL5A72002) {
      closedir(dir);
      return fd;
    }
    close(fd);
  }
  closedir(dir);
}
return -1;
```

```
...
pread(fd, &v, sizeof(v), REGISTER);
...
pwrite(fd, &v, sizeof(v), REGISTER);
```

Pilote en mode noyau :

```
static struct pci_device_id meye_pci_tbl[] = {
  { PCI_VENDOR_ID_KAWASAKI,
    PCI_DEVICE_ID_MCHIP_KL5A72002,
    PCI_ANY_ID, PCI_ANY_ID, 0, 0, 0 },
  { }
};

MODULE_DEVICE_TABLE(pci, meye_pci_tbl);

static struct pci_driver meye_driver = {
  name:           "meye",
  id_table:       meye_pci_tbl,
  probe:          meye_probe,
  remove:         meye_remove,
}

static int __init meye_init_module(void) {
  ...
  pci_module_init(&meye_driver);
  ...
}
```

```
...
pci_read_config_dword(pci_dev, REGISTER, &v);
...
pci_write_config_dword(pci_dev, REGISTER, v);
```

Changements dans l'accès à la mémoire PCI

Pilote en mode utilisateur :

```
void *mchip_base;
int fd;
u32 mem;
...
pci_config_read_u32(mchip_dev,
                    PCI_BASE_ADDRESS_0,
                    &mem);
...
fd = open("/dev/mem", O_RDWR|O_SYNC);
...
mchip_base = mmap(0, REGS_SIZE,
                     PROT_READ | PROT_WRITE,
                     MAP_SHARED, fd,
                     mem & ~(PAGE_SIZE - 1));
if (mem & (PAGE_SIZE-1))
  mchip_base += (mem & (PAGE_SIZE-1));
```

Pilote en mode noyau :

```
unsigned long mchip_adr;
unsigned char *mchip_base;
...
mchip_adr = pci_resource_start(mchip_dev,0);
...
mchip_base = ioremap(mchip_adr, REGS_SIZE);
```

Changements dans l'accès DMA

Pilote en mode utilisateur :

```
#define TOKEN 0xfeeb0000
int i;
void *vmem;
u32 ptable[npages];
int fd;
off_t addr;

fd = open("/dev/mem", O_RDONLY|O_SYNC);
...
vmem = memalign(PAGE_SIZE,
                (npages + 1) * PAGE_SIZE);
...
memset(vmem, 0, npages * PAGE_SIZE);
mlock(vmem, npages * PAGE_SIZE);
for (i = 0; i < npages; i++)
  *(u32 *)(vmem + i * PAGE_SIZE) = TOKEN | i;
for (addr = 0; addr < MAX_MEMORY;
     addr += PAGE_SIZE) {
  u32 v;
  pread(fd, &v, sizeof(v), addr);
  if ((v & 0xFFFF0000) != TOKEN) continue;
  i = (v & 0xFFFF);
  ...
  ptable[i] = addr;
}
close(fd):
```

Pilote en mode noyau :

```
static void *rvmalloc(signed long size) {
  void *mem;
  unsigned long adr, page;

  mem = vmalloc_32(size);
  if (mem) {
    memset(mem, 0, size);
    adr = (unsigned long)mem;
    while (size > 0) {
      page = kvirt_to_pa(adr);
      mem_map_reserve(virt_to_page(__va(page)));
      adr += PAGE_SIZE;
      size -= PAGE_SIZE;
    }
  }
  return mem;
}
...
int i = 0;
void *vmem;
u32 ptable[npages];
signed long size;

size = npages * PAGE_SIZE;
vmem = rvmalloc(size);
if (!vmem) return NULL;
memset(ptable, 0, sizeof(ptable));
adr = (unsigned long)vmem;
while (size > 0) {
  ptable[i++] =
    virt_to_bus(__va(kvirt_to_pa(adr)));
  adr += PAGE_SIZE;
  size -= PAGE_SIZE;
}
```

Changements dans l'attente active

Pilote en mode utilisateur :

```
...
while (inb(IOPORT) & 2) ;
...
```

Pilote en mode noyau :

```
#define wait_on_command(command) { \
  unsigned int n = 10000; \
  while (--n && (command)) \
    udelay(1); \
  if (!n) \
    printk(KERN_WARNING "command failed")
}
...
wait_on_command(inb_p(IOPORT) & 2);
...
```

Changements dans la gestion des interruptions

Pilote en mode utilisateur :

```
u8 v1, v2, ov1=0, ov2=1;
while (1) {
  v1 = inb(IOPORT1);
  v2 = inb(IOPORT2);
  if (v1 != ov1 || v2 != ov2)
    printf("event 0x%02x 0x%02x\n", v1, v2);
  ov1 = v1;
  ov2 = v2;
}
```

Pilote en mode noyau :

```
void sonypi_irq(int irq, void *dev_id,
                struct pt_regs *regs) {
  u8 v1, v2;
  v1 = inb(IOPORT1);
  v2 = inb(IOPORT2);
  printk(KERN_INFO "event 0x%02x 0x%02x\n",
         v1, v2);
  ...
}
...
request_irq(IRQ, sonypi_irq, SA_SHIRQ,
            "sonypi", sonypi_irq);
...
```

Pilotes matériels sous Linux :
mode noyau
versus
mode utilisateur

conclusion

Conclusion

- Décision importante lors du design
- Pas de ”recette universelle”
- Souvent, une combinaison des deux est la meilleure solution

Liens

- Alcôve http://www.alcove.com

- Alcôve-Labs http://www.alcove-labs.org

Retour

Fermer