www.lisyc.univ-brest.fr/pages_perso/babau/

Ecole Temps Réel

Conception orientée objet appliquée à l'embarqué et au temps réel

Logiciel pour la communication

avec un procédé via des périphériques

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire LISyC

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
- Implémentation des pilotes
  - Mise en œuvre dans les OS monolithiques (Linux)
  - Pilotes intégrés
    - Périphérique vu comme un fichier
  - Couplage matériel / logiciel
- Architecture des logiciels de communication
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
- Implémentation des pilotes
  - Mise en œuvre dans les OS monolithiques (Linux)
  - Pilotes intégrés
    - Périphérique vu comme un fichier
  - Couplage matériel / logiciel
- Architecture des logiciels de communication
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Pilote logiciel

- Objectif
  - Masquer les contraintes matérielles
  - Découpler l'application du matériel
  - Protéger/partager l'accès aux ressources matérielles
- Principes
  - Primitives d'initialisation / Terminaison
    - Initialisations (driver et périphérique, logiciel et matériel) (*insmod, mknod)*
    - Arrêt (driver et périphérique, logiciel et matériel) (*rmmod, rm*)
  - Primitives d'accès au périphérique
    - Connexion/ déconnexion (*open, close*)
  - Primitives de communication
    - Lecture/écriture (*read, write*)
  - Primitives de configuration
    - Paramètres de la communication (*ioctl*)
  - Pilote intégré
    - Utilisation du système d'entrée/sortie
    - Un périphérique = un fichier en lecture ou en écriture

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Exemples

- Pilotes existants
  - Gestion de l'accès à une liaison série
  - Gestion de l'accès à une imprimante
  - Gestionnaire de fichiers
- Liaison série
  - Driver série de périphériques
  - RS232 : voltage, CTS, RxD, TxD, adresse carte, …

jean-philippe.babau@univ-brest.fr

Exemple

```
char bufferE[20];  int fdE;

fdE=open("/dev/ttyS0",O_WRONLY);
/*ouverture d'une liaison série en ecriture */
if (fdE < 0)
     {      printf(« erreur connexion\n");
            exit (-1);
      }
else
     {      bufferE [0] = 1   ; bufferE[1] = 0   ; bufferE[2] = 2  ;
            bufferE [3] = 65 ; bufferE[4] = 66 ; bufferE[5] = 13 ;
            write(fdE,bufferE,6);  /* envoi trame */
     } ;

close(fdE); /*arrêt connexion*/
```

jean-philippe.babau@univ-brest.fr

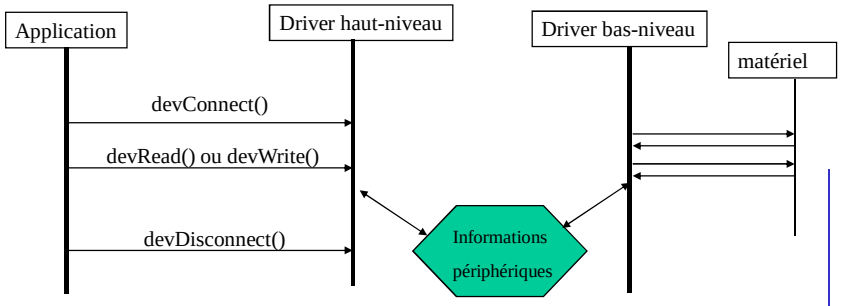
Exemple

```
char bufferR[20];  int fdR; int nCar; int nMax;

bufferR[0] = 0;
fdR=open("/dev/ttyS1",O_RDONLY);              /*ouverture d'une liaison série en lecture*/
if (fdR < 0)
     {      printf(« erreur connexion\n");
            exit (-1);
      }
else
     {      while(bufferR[0]!= 1)
            {           read(fdR, bufferR,1) ; }

            read(fdR, bufferR,1) ;
            if (bufferR[0]== 0)
            {           read(fdR, bufferR,1) ; nCar = buffeR[0];
                        nMax = read(fdR,bufferR,nCar) ;
                        read(fdR, bufferR,1) ;
                        if (bufferR[0]== 13)
                        {           close(fdR);   /*arrêt connexion*/
                                    exit(0) ;
                        }
            }
      printf(« erreur communication");
      close(fdR);
      exit (-2);
      }
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Principe général



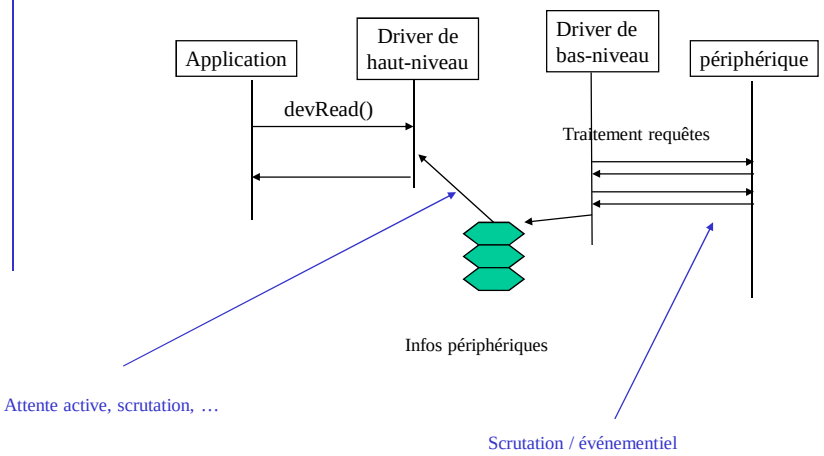
Ecole Temps Réel

Primitives de connexion

- Opération logicielle
  - Connexion aux services d'accès au périphérique
  - Pas d'impact sur le matériel
- Ouvrir un point de connexion (idem open pour un fichier)
  - *devDescriptor = devConnect(char* deviceName)*
  - Vérifications
    - Nombre de connexions simultanées
    - Type de connexion (read/write)
- Fermer un point de connexion
  - *devDisconnect(int devDescriptor) ;*
- Vérifications
  - Opérations en cours ?

Ecole Temps Réel

Lecture



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives de lecture

- Taille des informations
  - **Constant** ou variable
  - Nombre d'informations mémorisées
- Consommation
  - Attente active (Wait)
    - Fin de l'attente : nombre d'octets maximal / octet de fin / timeout
  - Scrutation (Read)
    - Lecture d'une ou de plusieurs informations
  - Consommation des informations
- Mémorisation
  - Zone de stockage
    - taille, type (FIFO, LIFO, autres)
  - Écrasement ou perte des informations
    - Mémorisation du dernier message
  - Datation des informations (durée de validité)
- Communication avec l'environnement
  - Scrutation périodique ou par interruption

jean-philippe.babau@univ-brest.fr

Exemples de primitives de lecture

- Scrutation

  *int readData (int devDescriptor , char * buffer)*

  Place dans *buffer* (pré-alloué pour une taille de 2 octets) la dernière valeur D envoyée par le périphérique P identifié par son jeton de connexion *devDescriptor*. Après 10 secondes, une donnée reçue n'est plus disponible pour l'application.

  Retour :
  - -1 : périphérique non connecté
  - -2 : pas de donnée disponible
  - nb : nombre d'octets reçu (2)

- Attente

  *int waitNextData (char * buffer, int timeout)*

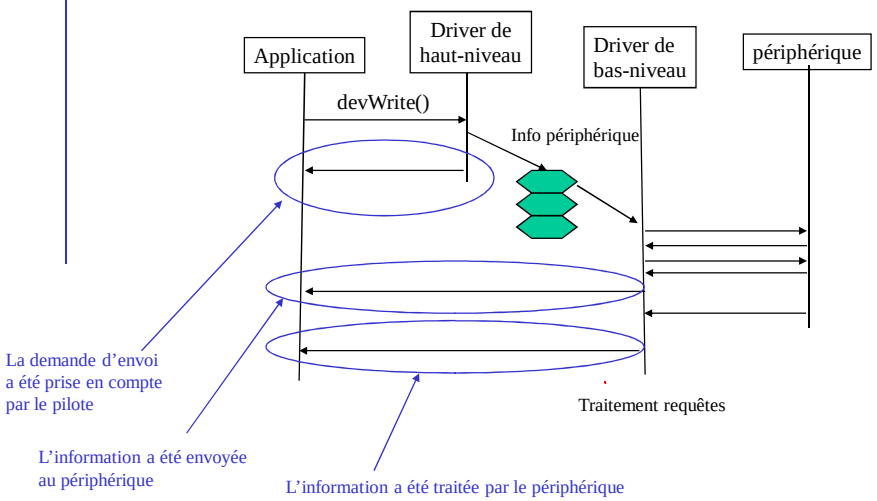
  Récupère le dernier message envoyé par le périphérique P. Le message est placé dans *buffer* (pré-alloué pour une taille de 4 octets) et consommé par l'application. Si aucun message n'est arrivé ou tous les messages ont déjà été consommé, attend (au maximum *timeout* secondes) le prochain message. Conserve les deux derniers messages.

  Retour :
  - -1 : périphérique non connecté
  - -2 : *timeout* expiré
  - nb : nombre d'octets reçu (4)

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Ecriture



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives d'écriture

- Taille des informations
  - Constant ou variable
- Envoi
  - Ecriture effective ou demande d'écriture
- Mémorisation
  - Zone de stockage
    - taille, type (FIFO, LIFO, autres)
  - Écrasement ou perte des informations
- Communication avec l'environnement
  - Émission a dates prédéfinies (périodique) ou événementielle
    - Activation régulière d'actionneurs
    - Attente de disponibilité du périphérique (prêt à émettre)
- Niveau d'accusé réception
  - Remplissage des buffers matériels

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Exemples de primitives d'écriture

- Envoi en différé

  *int writeData (int devDescriptor, char * buffer, int nbOctets)*

  Demande d'envoi des *nbOctets* octets contenues dans *buffer* sur le périphérique P identifié par son jeton de connexion *devDescriptor*. L'envoi est effectif au maximum dans les 100 ms.

  Retour :
  - -1 : périphérique non connecté
  - -2 : *nbOctets* est strictement supérieur à 8
  - -3 : demande rejetée car en cours d'émission
  - nb : les données ont été transférées et sont prêtes à être émises
- Envoi immédiat

  int writeData (int devDescriptor , int val)

  Envoie *val* sur le périphérique P identifié par son jeton de connexion *devDescriptor*

  Retour :
  - -1 : périphérique non connecté
  - -2 : *val* invalide (trop élevé)
  - nb : la donnée a été envoyé sur le périphérique

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives de configuration

- Paramètres de communication
  - Débit, adresses, …
- Type de communication
  - Stockage des informations, attente active, …
- Exemple
  - *status = ioctl(Serial_ID_1 , BAUDRATE, 9600) ;*

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

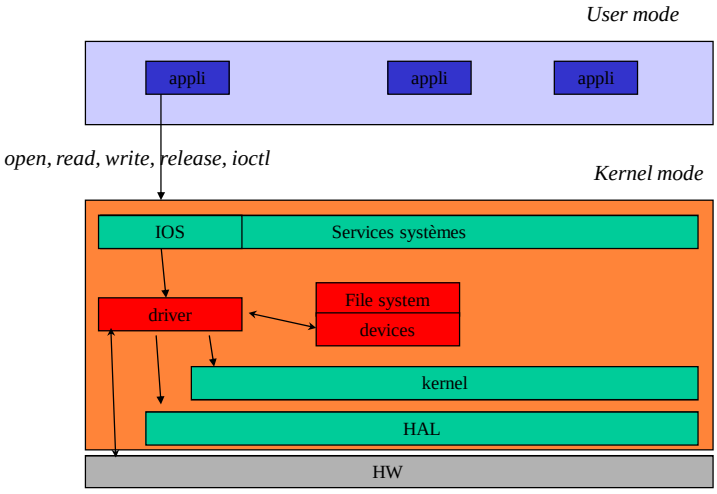
Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
- Implémentation des pilotes
  - Mise en œuvre dans les OS monolithiques (Linux)
  - Pilotes intégrés
    - Périphérique vu comme un fichier
  - Couplage matériel / logiciel
- Architecture des logiciels de communication
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

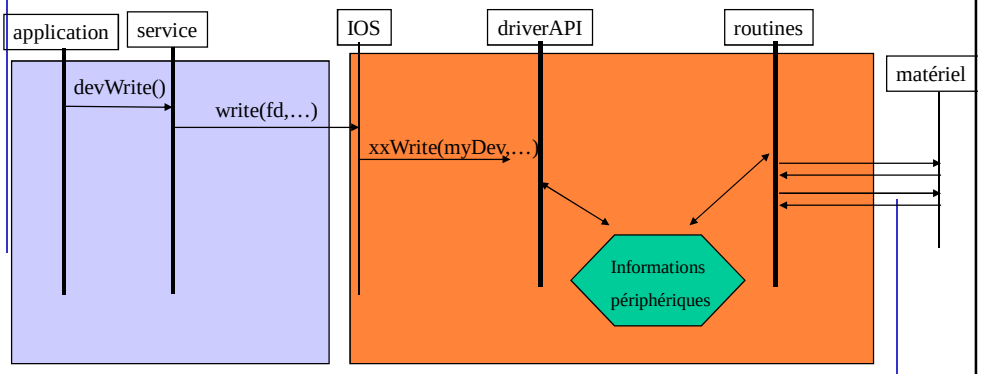
Architectures des systèmes



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Principes (POSIX)



- Appel de l'IOS synchrone ou asynchrone
- Logiciel dédié en mode kernel
  - lié au matériel, au SE

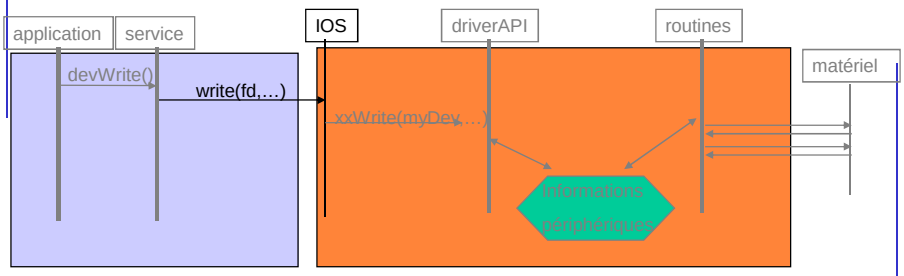
Logiciel à réaliser

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Pilote intégré
  - Appel de l'IOS
  - Appel du driver
  - Communication bas-niveau



Ecole Temps Réel

Primitives de l'IOS (*Linux*)

- Ouvrir
  - *fd = open ("name ", flag, mode) ;*
  - Connexion au périphérique *name*
  - fd : file descriptor (≠ 0,1,2) si connexion ok, ERROR sinon
  - flag : O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC
  - mode : 06444 sous unix
- Fermer
  - *close (fd) ;*
  - Déconnexion du périphérique fd

Ecole Temps Réel

Primitives de l'IOS (*Linux*)

- Lire
  - *nBytes = read (fd,&buffer,maxBytes) ;*
  - Lecture de *maxBytes* caractères sur *fd* et stockage dans *buffer*
  - maxBytes : nombre maximum d'octets à lire
  - nBytes = nombre d'octets lus (-1 : erreur)
  - ERROR : non ouvert, pas de xxRead
- Ecrire
  - actualBytes = *write (fd,&buffer,maxBytes) ;*
  - Écriture des *maxBytes* caractères de *buffer* sur *fd*
  - maxBytes : nombre d'octets à écrire
  - actuelBytes : nombre d'octets écrits (si ≠ maxBytes : erreur)
  - ERROR : non ouvert, pas de xxWrite

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives de l'IOS (*Linux*)

- Autre
  - *result = ioctl (fd, function,arg) ;*
  - Configuration de *fd* selon *function* et *arg*
  - function : code
  - arg : paramètre
  - exemple

```
#include <termios.h>
#define BAUDRATE B9600
int fd;
struct termios tio;
fd = open("/dev/ttyS0", O_RDONLY);
tio.c_cflag = BAUDRATE;
iotcl(fd, TCSETSF, 0);    // flush,  idem tcflush (fd, TCIFLUSH);
iotcl(fd, TCSETS,&tio);
// mise à jour du baudrate , utilisation équivalente à tcsetattr( fd, TCSANOW ,&tio);  : préférable
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives asynchrones de l'IOS

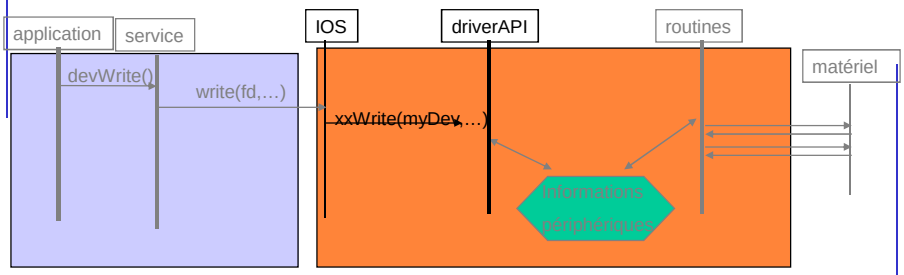
- Désynchronisation application / pilote
  - Communication par signal
    - Driver : émission du signal SIGIO
    - Application : attente sur signal SIGIO
  - Mise en place d'un *read* bloquant
    - Attente de SIGIO, puis appel à *read*
  - Attente avant une ré-écriture
    - Attente de SIGIO, puis appel à *write*

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Pilote intégré
  - Appel de l'IOS
  - Appel du driver
  - Communication bas-niveau



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

L' I/O system

- Installations des drivers et des périphériques
  - Installation / désinstallation d'un pilote : installation/désinstallation d'un module noyau
    - Attribution d'un numéro : le majeur
  - installation / désinstallation de périphériques : ajout/suppression d'un fichier dans /dev/
    - Pour un pilote identifié par son majeur
    - Attribution d'un numéro : le mineur

- Gestion des opérations sur les périphériques (open,close,read,write,ioctl)
  - API correspondante du driver
    - myOpen, myRelease, myRead, myWrite, myIoctl
  - L'appel du driver est réalisé par l'IOS (**transparente pour l'utilisateur)**
    - recherche du pilote par son numéro majeur
    - appel de la primitive correspondante du pilote
      - Lien open / myOpen, …

- Un périphérique est vu comme un fichier, géré par UN driver

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Installation d'un module

- Module noyau
  - Macros module_init et module_exit
  - Primitives __init et __exit

```
#include <linux/module.h> // needed for modules
#include <linux/kernel.h> // needed for KERN_ALERT
#include <linux/init.h> // needed for macro
MODULE_AUTHOR("JP Babau");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("premier driver");

static int __init install(void)
{    printk(KERN_ALERT "hello\n");
     return 0; }

static void __exit desinstall(void)
{    printk(KERN_ALERT "see you\n"); }

module_init(install);
module_exit(desinstall);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Installation d'un module

- Compilation spécifique

```
obj-m += unDriver.o
default :
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean :
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- Installation
  - Chargement de module noyau
    - *insmod Pilote.ko :* appel de *install*
  - Déchargement de module noyau
    - *rmmod Pilote :* appel de *desinstall*
  - Liste des modules chargés
    - *lsmod ou cat /proc/modules*

```
#!/bin/sh
driverModule="unDriver"
rmmod ${driverModule}.ko
insmod ${driverModule}.ko
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Installation d'un driver

- Enregistrement du driver

int = register_chrdev(unsigned char myMajorNumber, char * drvName, struct file_operations * fops);

  - Attribution dynamique d'un numéro majeur si myMajorNumber est à 0
    - register_chrdev() renvoie le numéro attribué
  - Attribution statique d'un numéro majeur sinon
    - Utilisation d'un périphérique indépendante de l'installation
  - Liste des numéros majeurs attribués
    - Assignés : 0-230; non assignés 231-239; local use : 240-254
    - cat /proc/devices

unDriver.h

```
#define DRIVERNAME "myDriver"
```

unDriver.c

```
static int myMajor = 0;   // allocation dynamique
static int __init install(void)
{    int installCR; …
     installCR = register_chrdev(myMajor, DRIVERNAME, &fops);
     if (installCR <0)
     {       printk(KERN_WARNING "probleme d'installation\n");
             return installCR;
     }
myMajor = installCR; … }
```

jean-philippe.babau@univ-brest.fr

Installation d'un driver

- Lien API IOS / API driver
  - Association primitives IOS / primitives pilote
  - Structure *file_operations*

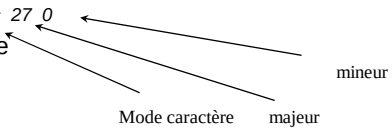
```
static struct file_operations fops =
{       open : myOpen,
        read : myRead,
        write : myWrite,
        ioctl : myIoctl,
        release : myRelease
};
```

- Désenregistrement du driver

```
int = unregister_chrdev(unsigned char myMajorNumber, char * drvName);
```


Gestion d'un périphérique

- Fichier
  - Identifié par un nom de fichier, un numéro de majeur et un numéro de mineur
  - Droits d'écriture
- Ajout d'un périphérique
  - *mknod /dev/ myDevice c 27 0*
- Retrait d'un périphérique
  - *rm /dev/ periph*
- Lister les périphérique
  - *ls /dev*




```
#!/bin/sh
driverName="myDriver"
deviceName="myDevice"
modeDevice="a+w"
rm -f /dev/${deviceName}[0-1]
majorNumber=$(awk '$2~/myDriver/ {print $1}' /proc/devices)
mknod /dev/${deviceName}0 c $majorNumber 0
mknod /dev/${deviceName}1 c $majorNumber 1
chmod ${modeDevice} /dev/${deviceName}[0-1]
```

API du pilote pour la gestion des périphériques

- Appelée par l'IOS
  - une par appel de l'IOS
- Paramètres Linux
  - pointeur vers le descripteur du périphérique (*inode* ou *file*)
  - paramètres de l'appel de l'IOS (*buf* et *count*)

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

int  myRelease(struct inode * inode, struct file * file);

ssize_t myRead (struct file * file, char* buf, size_t count, loff_t *ppos) ;

ssize_t myWrite (struct file * file, const char* buf, size_t count, loff_t *ppos) ;

int myIOctl (struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg);
```

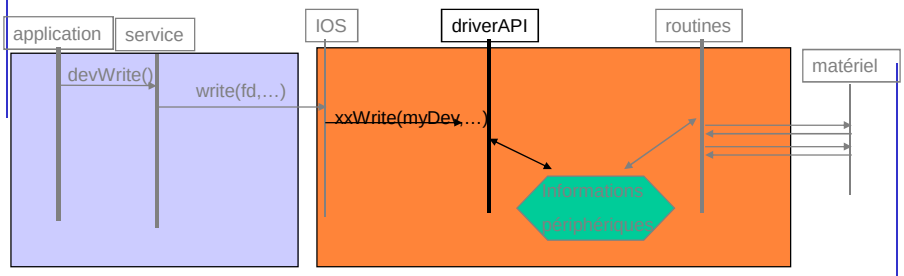

Vérifications effectuées

- IOS
  - Nom du périphérique
  - Connexion
    - pas de read avant open
  - Nombre de connexions
  - Type de connexion
    - pas de write sur O_RDONLY (non contrôlé par l'IOS de VxWorks)
- Driver
  - Ce qui n'est pas implémenté par l'IOS
  - Ce qui est spécifique au pilote
    - Par exemple : maxByte est trop important pour le périphérique concerné

Ecole Temps Réel

Plan

- Pilote intégré
  - Appel de l'IOS
  - Appel du driver
  - Communication bas-niveau

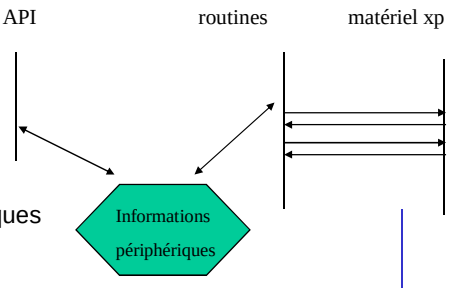


jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Informations gérées par le pilote

- Vue du périphérique
  - État du périphérique
  - Informations reçues
  - Informations datées
- Informations à destination du périphériques
  - Données à mémoriser
  - Données à afficher, imprimer, …
  - Commandes à réaliser
- Le pilote gère et manipule une vue du périphérique
  - Initialisation (adresse, …)
  - Configuration : ioctl
  - Connexion : open/close ou creat/remove
  - Lecture / écriture : read / write



jean-philippe.babau@univ-brest.fr

Gestion des données périphériques

- Nombre maximal de périphériques
- Type de données spécifiques au périphérique
  - Autres que celles gérées par l'IOS
    - Pas de champ name, indice, …

unDriver.h

```
#define MAXDEVICES 16

typedef struct
{
  int value;    // donnée périphérique
  int cpt;      // nombre de mises à jour
} devInfo;
```


Initialisation des données via l'installation

```
devInfo theDevices[MAXDEVICES];

static int __init install(void)
{
      int installCR;

      installCR = register_chrdev(myMajor,DRIVERNAME,&fops);
      myMajor = installCR;

      // initialisation des informations périphériques
      for (i=0;i<MAXDEVICES;i++)
      {
            theDevices[i].cpt = 0;
            theDevices[i].value = -1;
      }

      return 0;
}
```

Ecole Temps Réel

Association informations des périphériques – file system

- Assurer un lien avec les données gérées par l'IOS
  - Initialiser le champ *private_data* de la structure *file*
  - Utilisation du mineur comme indice
    - Si plusieurs périphériques

```
static ssize_t myOpen(struct inode *inode, struct file* file)
{
        // récupération du mineur
        unsigned  int minor  = iminor(inode);
        if ((minor>=0) && (minor<MAXDEVICES) )
        {
        // association descripteur périphérique / données spécifiques
                file->private_data = (devInfo*) &theDevices[minor];
        return 0;
        } else
        { return -1; }
}
```


Ecole Temps Réel

Désinstallation et release

- Désenregistrement du driver

```
static void __exit desinstall(void)
{
        printk(KERN_ALERT "see you\n");

        unregister_chrdev(myMajor,DRIVERNAME);

        printk(KERN_DEBUG "desinstallation OK\n");
}


static ssize_t myRelease(struct inode *inode, struct file* file)
{
        return 0;
}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Opération de transfert : écriture

- Copie de données du mode *user* au mode *kernel*
  - Primitive spécifique de copie

unsigned long copy_from_user (void * to, const void * from, unsigned long nbBytes);

- Renvoie le nombre d'octets écrits

```
#include <asm/uaccess.h>
static ssize_t myWrite(struct file *file, const char * buf, size_t count, loff_t *ppos)
{
        devInfo * p = (devInfo*) file->private_data;

        // sauvegarde des données périphériques
        copy_from_user((int*)&(p->value), buf, sizeof(int));

        p->cpt++;

        return (sizeof(int));
}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Opération de transfert : lecture

- Copie de données du mode *kernel* au mode *user*
  - Primitive spécifique de copie

unsigned long copy_to_user (void * *to*, const void * *from*, unsigned long *maxBytes*);

- Renvoie le nombre d'octets lus

```
#include <asm/uaccess.h>
static ssize_t myRead(struct file *file, char * buf, size_t count, loff_t *ppos)
{
        devInfo * p = (devInfo*) file->private_data;

        // récupération des données spécifiques
        copy_to_user(buf, (devInfo*) p, sizeof(devInfo));

        return (sizeof(devInfo));
}
```

jean-philippe.babau@univ-brest.fr

Primitive ioctl

- Primitive spécifique
  - Utlisation normale en configuration

    int myIOctl (struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg);

    - *cmd* : numéro de la commande à exécuter
    - *arg* : argument de la commande à exécuter
  - Utilisation en lecture ou écriture
    - cast de *arg*, utilisé comme une adresse de buffer

- Structure type de la fonction ioctl

```
int myIoctl(struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg)
{
        int ret=0;

        switch(cmd)
        {       case CMD0 : …;
                case CMD1 : …;
                default : ret = EINVAL; break;
        }
        return (ret);
}
```

jean-philippe.babau@univ-brest.fr

Primitive ioctl

- Définition de cmd
  - Numéros préréservés dans le système
  - Choisir un numéro
    - Choisir un nombre magique magicNumber (de type char)
    - Construire le numéro à partir d'une macro
      - Configuration : _IO(char magicNumber, int value)
      - Lecture : _IOR(char magicNumber, int cmdValue, type)
      - Écriture : _IOW(char magicNumber, int cmdValue, type)

unDriver.h

```
#include <linux/ioctl.h>

#define myDRIVER_IO_MAGIC  'd'

#define RAZ  _IO(myDRIVER_IO_MAGIC, 1)
define GETVAL _IOR(myDRIVER_IO_MAGIC, 2, int)
#define SETVAL _IOW(myDRIVER_IO_MAGIC, 3, int)
```

jean-philippe.babau@univ-brest.fr

Primitive ioctl

- Lectures et écritures
  - Copie de données entre mode *kernel* et mode *user*
    - Primitives spécifiques de copie si *arg* est considéré comme l'adresse d'un buffer
- Renvoie si la commande a été correctement effectuée

```
#include <linux/fs.h>

static int myIoctl(struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg)
{
        devInfo * p = (devInfo*) file->private_data;
        int ret=0;
        switch(cmd)
        {       case RAZ : p->cpt = 0;break;
                case GETVAL : copy_to_user((int*)arg, (int*) & (p->value), sizeof(int)); break;
                case SETVAL : p->value = (int) arg ; break;
                default : ret = EINVAL;break;
        }
        return (ret);
}
```

jean-philippe.babau@univ-brest.fr

Appel du pilote

```
#include "unDriver.h"

int main(void)
{
int devID;

int *pWrite = (int*) malloc(sizeof(int));

*pWrite = 1;
devID = open("/dev/myDevice3", O_RDWR);

while (1)
{
sleep(2);
write( devID, (char*)pWrite, sizeof(int));

*pWrite += 3;
}

close(devID);
}
```

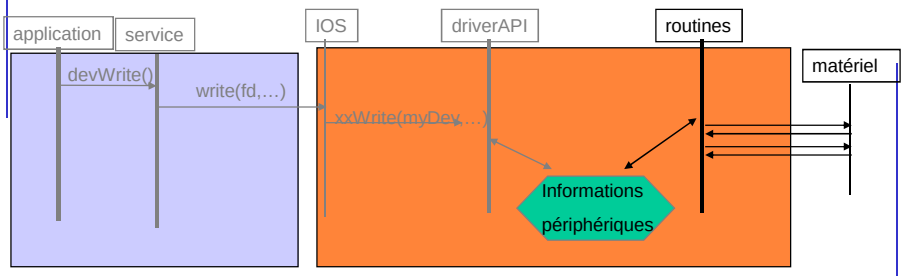
```
#include "unDriver.h"

int main(void)
{

devInfo *pRead = (devInfo*) malloc(sizeof(devInfo));

int redBytes;
int devID = open ("/dev/myDevice3", O_RDONLY);

redBytes= read(devID,(char*)pRead,sizeof(devInfo));

printf("lecture de %i(%i)sur %i octets\n",pRead->value,pRead->cpt,redBytes);

ioctl(devID,RAZ,0);

redBytes= read(devID,(char*)pRead,sizeof(devInfo));

printf("lecture de %i(%i)sur %i octets\n",pRead->value,pRead->cpt,redBytes);

close(devID);

}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Pilote intégré
  - Appel de l'IOS
  - Appel du driver
  - Communication bas-niveau

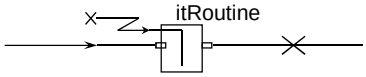


jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

L'interruption

Création



Destruction

C/POSIX

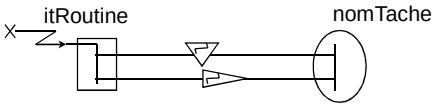
```
#include <linux/interrupt.h>
static irqreturn_t itRoutine(int irq, void* devID)
{ …
return IRQ_HANDLED;}
…
request_irq(itNumber, itRoutine, flag, driverName, listeDevID) ;
// flag : SA_INTERRUPT ou SA_SHIRQ pour une interruption partagée par plusieurs devices
free_irq(itNumber, xx);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

L'interruption

Masquage / démasquage



C/POSIX

```
enable_irq(int itNumber) ;
disable_irq(int itNumber) ;
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

timer

- Gestion en temps absolu
  - Réarmement nécessaire pour un timer périodique
  - Unité de temps (unsigned long) : *jiffies (ms)*

```
#include <linux/timer.h>
#define TIMERPERIOD 1250
struct timer_list myTimer;
int counter;
void periodicCounter(unsigned long arg)
{
        spin_lock_irq(&lockCounter);
                counter++;
        spin_unlock_irq(&lockCounter);
        myTimer.expires += (unsigned long) TIMERPERIOD;
        add_timer(&myTimer);
}
int __init install(void)
{ …
        init_timer(&myTimer);
        myTimer.expires = jiffies + (unsigned long) TIMERPERIOD;
        myTimer.function = periodicCounter;
        counter = 0;
        add_timer(&myTimer); …
```

jean-philippe.babau@univ-brest.fr

Le sémaphore en mode noyau

- Principes
  - Sémaphore à compte
  - Utilisable en mode mutex

```
struct semaphore sem;
sema_init(& sem, initVal);  ou init_MUTEX(&sem); ou init_MUTEX_LOCKED(&sem);

up(& sem);                  // valeur courante du sémaphore ++

down(& sem);                // attend que la valeur courante du sémaphore>0, puis --
down_interruptible(& sem);  // idem down, interruptible : si IT, pas de prise de sémaphore
down_trylock (& sem);       // prend sem si la valeur est positive, retourne 0 sinon
```


Spinlock

- Principes
  - Principe du mutex
  - Utilisable dans une routine

```
spinlock_t myLock;
spin_lock_init(& myLock);  ou myLock = SPIN_LOCK_UNLOCKED;

spin_lock(& myLock);        // prend myLock, ou attend si il est déjà pris

spin_unlock(& myLock);      // libere myLock
```

Ecole Temps Réel

Couplage avec le matériel

- Accès au matériel via les ports
  - Lecture

```
byte valB;
word valW;
long valL;
valB = inb(adPort);
valW = inw(adPort);
valL = inl(adPort);
```

  - Écriture

```
outb(valB, adPort);
outw(valW, adPort);
outl(valL, adPort);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives spécifiques mode noyau

- Gestion mémoire

```
#include <linux/malloc.h>
void * kmalloc (size_t taille, int priorité);
// Priorités : GFP_KERNEL, GFP_USER, GFP_ATOMIC
void kfree (const void *ptr);
```

- Gestion de l'heure

```
struct timeval timeZero;
do_gettimeofday(&timeZero);
```

- Debug et suivi

```
printk(KERN_ALERT "attention : %i\n" , i);
// utilisation possible de KERN_DEBUG, KERN_ALERT, KERN_ERR, KERN_INFO
```

  - Visualisation des messages en mode commande
    - appel à *dmesg*

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
- Implémentation des pilotes
  - Couplage matériel / logiciel
  - Pilotes intégrés
    - Périphérique vu comme un fichier
  - Mise en œuvre dans les OS classiques (Linux)
- Architecture des logiciels de communication
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Primitives d'initialisation / terminaison

- Initialisation matérielle
  - Configuration du système (attention aux effets de bord)
    - Port, horloge, …
  - Configuration du matériel
  - Séquence d'initialisation
- Initialisation logicielle
  - Déclaration et allocation des buffers
    - Taille, type
  - Mise en place des éléments de communication
    - Sémaphores, mutex
  - Activation des interruptions, alarmes, timers, …
- Arrêt
  - Logiciel : libération mémoire, masquage des interruptions, arrêt des timers
  - Matériel : désactivation des composants inutilisés

jean-philippe.babau@univ-brest.fr

Gestion des éléments

- Création/initialisation, destruction
  - Statique
    - Lors de l'installation et de la désinstallation
  - Dynamique : à éviter
    - Gestion de structure par périphérique
- Gestion des erreurs
  - Pilote inexistant, déjà installé
  - Périphérique déjà créé, …
  - Attente infinie de données
  - Données invalides

Architecture en couches

- Driver de bas niveau
  - Contrôle de la carte, du périphérique
- Driver de haut niveau
  - Initialisation, connexion, lecture, écriture, configuration
  - Virtualisation des périphériques
- Couche service
  - Administration
  - Partage et droit d'accès
  - Services évolués (multi-périphériques)

Architecture en couches

- Driver de driver …
- Lecteur code barre accessible via une liaison série
  - pilote de liaison série
  - pilote du lecteur code barre
- Périphérique en réseau
  - pilote du réseau
  - pilote de périphériques

La couche service

- Interconnexion application / pilote
  - Service orienté application
    - abstraction de l'IOS
    - Initialisations et appels implicites à l'IOS

```
int  GetSpeed(int v)
      {  int fdABS;
         int maxBytes = 8;
         int * buffer = (int*) malloc(maxBytes * sizeof(char) );
        fdABS= open(" DeviceName ",O_RDONLY);
         read ( fdABS,&buffer, maxBytes) ;
         return (*buffer);
      }
```

  - protocole de communication application / pilote
    - Appel bloquant d'une primitive non-bloquante

Ecole Temps Réel

La couche service

- Gestion du service de communication
  - Administration
    - Installation / libération / réinitialisation
    - Configuration
  - Suivi
  - Utilisateur
    - Droit et temps d'accès
    - Partage : mono / multi utilisateur
      - Protection des données
    - Définir la notion d'utilisateur : tâche ou application ou …

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La couche service

- Fonctions avancées
  - accès multiples
    - open( ); open( );
  - envois multiples
    - write( ); write( )
  - envois périodique
    - onPeriodicAlarm { sendOnBoolSem(); }
    - periodicTask { while(1) { waitOnBoolSem(); takeMutex(); write(); releaseMutex(); }}
  - échanges entre périphérique
    - read( ); write() /* échanger */

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Exemple de couche service : le joystick sous Windows

UINT joyGetNumDevs(VOID)

The joyGetNumDevs function returns the number of joysticks supported by the current driver or zero if no driver is installed.

MMRESULT joyGetPos(UINT *uJoyID*, LPJOYINFO *pji*)

uJoyID Identifier of the joystick to be queried. Valid values for *uJoyID* range from zero (JOYSTICKID1) to 15

pji Pointer to a JOYINFO structure that contains the position and button status of the joystick.

Returns JOYERR_NOERROR if successful or one of the following error values.

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Exemple de couche service : le joystick sous Windows

MMRESULT joySetCapture(HWND *hwnd*, UINT *uJoyID*, UINT *uPeriod*, BOOL *fChanged*)

hwnd Handle to the window to receive the joystick messages.

uJoyID Identifier of the joystick to be captured. Valid values for *uJoyID* range from zero (JOYSTICKID1) to 15

uPeriod Polling frequency, in milliseconds.

fChanged Change position flag. Specify TRUE for this parameter to send messages only when the position changes by a value greater than the joystick movement threshold. Otherwise, messages are sent at the polling frequency specified in *uPeriod*.

Returns JOYERR_NOERROR if successful or one of the following error values.

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Périphérique virtuel

- Définition de périphériques virtuels
  - Élément de l' IHM (souris, clavier, écran)
  - VirtualMouse de Windows
- Pilote de périphériques virtuels
  - Livré avec l'OS
  - Couche service adaptée
    - onClick()
  - Primitives du pilote inaccessibles
  - ? Pilote ?
- Pilote réel
  - Réalisation du pilote bas niveau
  - Interconnexion avec le périphérique virtuel
  - Émulation du périphérique virtuel

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Périphérique virtuel

- Inconvénients
  - Limitation du comportement
    - Pas de triple click
  - QoS
- Avantages
  - Développement rapide de pilotes
  - Applications indépendantes des périphériques
  - Sécurité

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Périphériques « distants »

- Réseau de périphériques
  - Réseaux de capteurs, réseaux d'appareils, périphériques sans fil
- Profils de haut niveau intégrés dans les protocoles
  - Surcouche aux protocoles de transport
  - Périphériques virtuels
  - Spécification génériques
    - Nom, adresse, liste d'opérations ou de services
- Profils Bluetooth
  - Fax Profile (FAX) : profil de télécopieur
  - Headset Profile (HSP) : profil d'oreillette
  - Serial Port Profile (SPP) : profil de port série
- Le protocole avec ses profils remplace le driver
  - Échanges standardisés
  - Communication (appareil photo, téléphone, imprimante) <-> ordinateur

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Conclusion

- Un pilote gère des périphériques tels des fichiers
- Un périphérique est géré par un seul pilote
- IOS : couche intermédiaire standardisée application/pilotes
- Pilote
  - Architecture dédiée : OS / matériel
  - Gestion d'une vue du périphérique
    - Correspondance avec les actions sur le périphérique
  - Architecture logicielle
    - Politique de stockage/consommation des informations
    - Concurrence
    - Statique / dynamique
    - Architecture en couches
- Couche service : services de haut niveau

jean-philippe.babau@univ-brest.fr

Références utiles et utilisées pour ce cours

- *broux.ftp-developpez.com/articles/c/driver-c-linux.pdf*
- J. Corbet, A. Rubini, G. Kroah-Hartman “LINUX DEVICE DRIVERS” O’Reilly Media, ISBN-13 : 978-0-596-00590-0, 2005

jean-philippe.babau@univ-brest.fr