lab-sticc.univ-brest.fr/~babau/

UBO

Logiciels de communication avec des périphériques

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

jean-philippe.babau@univ-brest.fr

UBO

jean-philippe.babau@univ-brest.fr

UBO

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
  - *Communication d'informations (cf. protocoles réseau)*
- 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
  - Allocation
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

UBO

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
  - *Communication d'informations (cf. protocoles réseau)*
- 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
  - Allocation
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

Exemples robot lego ® NXT

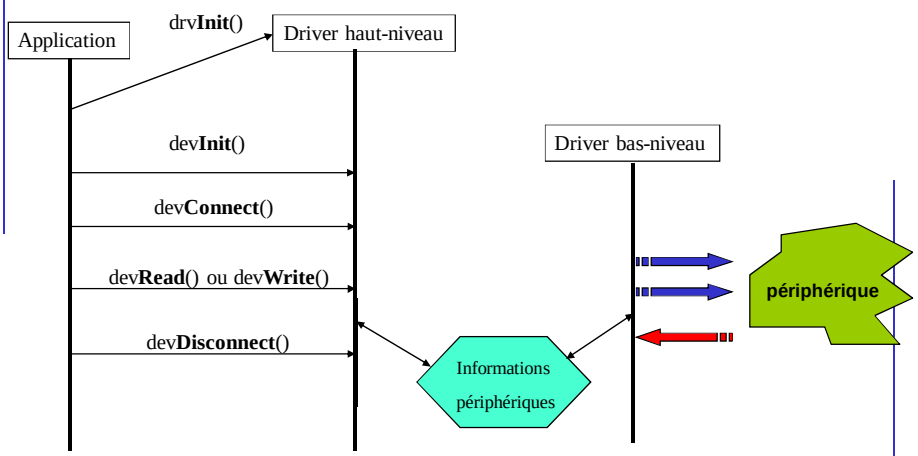
- Driver de capteur de luminosité
  - Primitive d'initialisation
    ```
    SetSensorLight(S1);
    ```
  - Primitive de lecture
    ```
    x = Sensor(S1);
    ```
  - Primitive de configuration
    ```
    SetSensorMode(S1, SENSOR_MODE_PERCENT);      // valeur entre 0 et 100
    SetSensorMode(S1, SENSOR_MODE_RAW);          // valeur entre 0 et 1023
    ```

- Driver de fichiers
  - Primitive d'initialisation
    ```
    CreateFile(string fileName, short size, byte handle);
    ```
  - Primitives de connexion
    ```
    OpenFileAppend(string fileName, short size, byte handle);
    CloseFile(byte handle);
    ```
  - Primitives de lecture
    ```
    ReadBytes(handle, array, nbOctets);
    ```

jean-philippe.babau@univ-brest.fr

UBO

Principe général



jean-philippe.babau@univ-brest.fr

UBO

Pilote logiciel

- Objectif
  - masquer les contraintes matérielles
    - Adresses, protocoles, é
  - découpler lôapplication du matériel
  - protéger/partager lôaccès à un périphérique

jean-philippe.babau@univ-brest.fr

UBO

Pilote logiciel

- Principes
  - Primitives dôinitialisation
    - Initialisations (driver et périphériques, logiciel et matériel)
    - Arrêt (driver et périphériques, logiciel et matériel)
  - Primitives dôaccès au service
    - Connexion/ déconnexion
  - Primitives de communication
    - Lecture/écriture de données ou dôenvoi/réception de messages
  - Primitives de configuration
    - Logiciel et matériel
    - Paramètres de communication avec le périphérique
  - Pilote intégré
    - utilisation du système dôentrée/sortie (IOS)
    - un périphérique : un fichier en lecture ou écriture

jean-philippe.babau@univ-brest.fr

UBO

Exemples

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

jean-philippe.babau@univ-brest.fr

UBO

Primitives dôinitialisation

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

jean-philippe.babau@univ-brest.fr

UBO

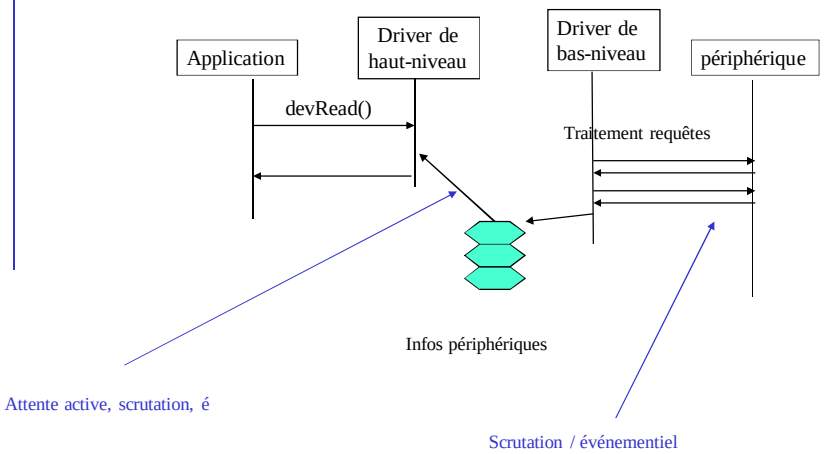
Primitives de connexion

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

jean-philippe.babau@univ-brest.fr

UBO

Primitive de lecture



jean-philippe.babau@univ-brest.fr

UBO

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

UBO

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)*

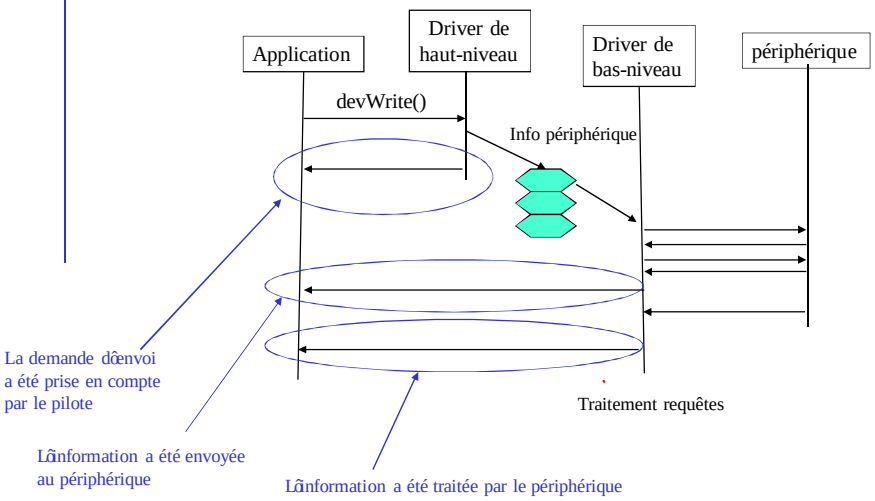
  Récupère le dernier message envoyé par le périphérique P. Le message est placé dans *buffer* (pré-alloué pour une taille de 4 octets) et consommé par l'application. Si aucun message n'est arrivé ou tous les messages ont déjà été consommé, attend (au maximum *timeout* secondes) le prochain message. Conserve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

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

jean-philippe.babau@univ-brest.fr

UBO

Primitive d'écriture



jean-philippe.babau@univ-brest.fr

UBO

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

UBO

Exemples de primitives dœcriture

- Envoi en différé

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

  Demande dœnvoi des *nbOctets* octets contenues dans *buffer* sur le périphérique P identifié par son jeton de connexion *devDescriptor*. Lœnvoi 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 dœmission
  - 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

UBO

Primitives de configuration

- Paramètres de communication
  - Débit, adresses, é
- Type de communication
  - Stockage des informations, attente active, é
- Exemple
  - *Serial_ID_1 = creat(« COM1 »,* O_RDONLY*)*
  - *status = ioctl(Serial_ID_1 , BAUDRATE, 9600) ;*

jean-philippe.babau@univ-brest.fr

UBO

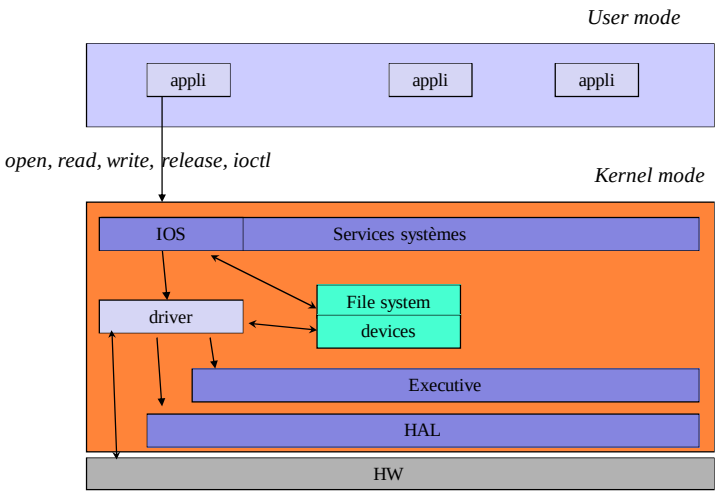
Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec l'environnement
  - *Communication d'informations (cf. protocoles réseau)*
- 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
  - Allocation
  - Architecture en couches
  - Gestion de services
    - Politique d'accès et de partage des services

jean-philippe.babau@univ-brest.fr

UBO

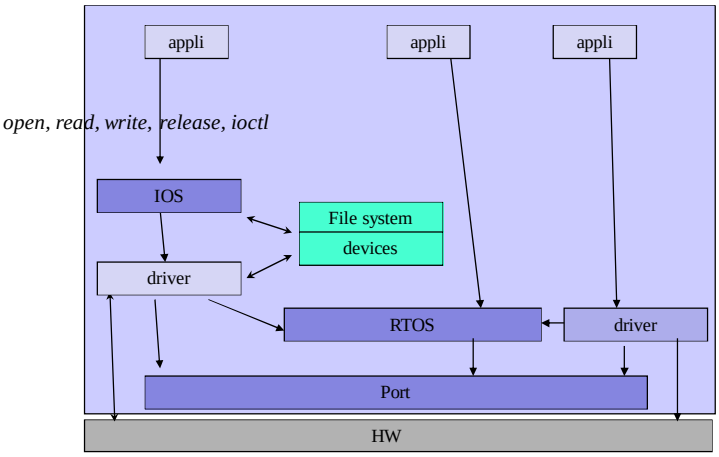
Architectures des systèmes monolithiques



jean-philippe.babau@univ-brest.fr

UBO

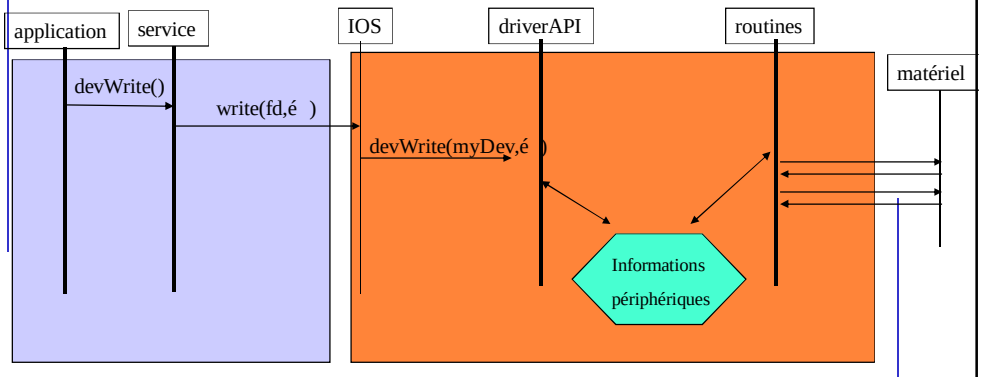
Architectures des systèmes temps réel



jean-philippe.babau@univ-brest.fr

UBO

Principes (POSIX)



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

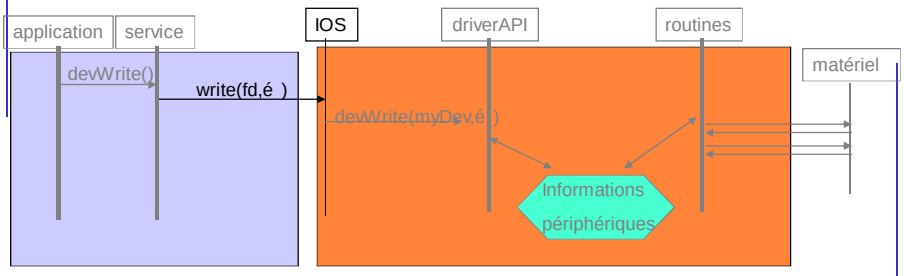
Logiciel à réaliser

jean-philippe.babau@univ-brest.fr

UBO

Plan

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



jean-philippe.babau@univ-brest.fr

UBO

Primitives de l'OS (*POSIX*)

- Ouvrir
  - *fd = open ("name ", flag, mode) ;*
  - Connexion au périphérique *name*
  - fd : file descriptor ($\neq$ 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

jean-philippe.babau@univ-brest.fr

UBO

Primitives de l'OS (*POSIX*)

- 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 devRead
- 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 devWrite

jean-philippe.babau@univ-brest.fr

UBO

Primitives de l'OS (*POSIX*)

- 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

UBO

Primitives asynchrones de l'OS (VxWorks)

- Découplage application / pilote
  - Émission non bloquante d'une requête
- Création d'une structure par requête
  - Paramètres de l'appel (idem paramètres de l'appel synchrone)
  - Etat de la requête (en-cours, terminée, é)
  - Exemple pour un appel asynchrone de read :
    - char buffer[TAILLE] ;
    - aiocb_read.aio_fildes = fd ;
    - aiocb_read.aio_buf = buffer ;
    - aiocb_read.aio_nbytes = TAILLE ;
    - etc.

jean-philippe.babau@univ-brest.fr

UBO

Primitives asynchrones de l'OS (VxWorks)

- Primitives
  - aio_read (& aiocb_read)
  - aio_write(& aiocb_write)
  - aio_return(& aiocb_write)
  - une utilisation simultanée d'une structure
- Suivi de la requête
  - (aio_error (& aiocb_read) = = EINPROGRESS)
  - aio_suspend(&aiocb[], nReq, timeout)
  - aio_cancel(&aiocb)
  - attente sur signal (cf. POSIX)

jean-philippe.babau@univ-brest.fr

UBO

Primitives asynchrones de lôOS (POSIX)

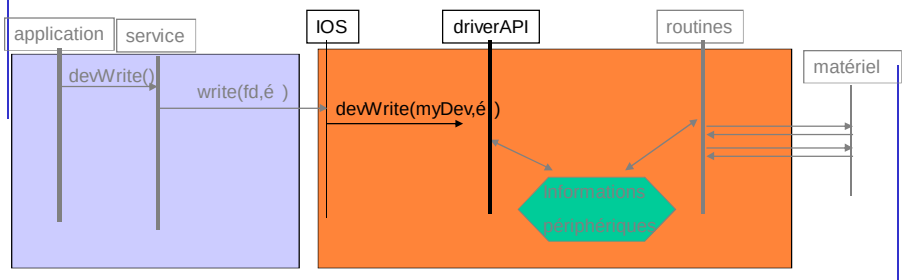
- Communication par signal
  - Driver : émission du signal SIGIO
  - Application : attente sur signal SIGIO
- Mise en place dôun *read* bloquant
  - Attente de SIGIO, puis appel à *read*
- Attente avant une ré-écriture
  - Attente de SIGIO, puis appel à *write*

jean-philippe.babau@univ-brest.fr

UBO

Plan

- Pilote intégré
  - Appel de lôOS
  - Appel du driver
  - Communication bas-niveau

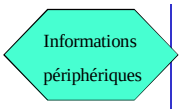


jean-philippe.babau@univ-brest.fr

UBO

Lôl/O system (IOS)

- Service de l'OS
  - Mise en î uvre **spécifique** à lôOS
- Couche intermédiaire entre lôapplication et les pilotes
- Un périphérique (au sens du logiciel) est vu comme un fichier, géré par **UN** driver
  - Un périphérique au sens du logiciel = une structure de donnée
  - En lien avec le périphérique réel
  - Plusieurs périphériques au sens logiciel pour un périphérique matériel
    - Attention aux conflits dôaccès é



- Installations des services
  - installation / désinstallation dôun pilote
  - installation / désinstallation de périphériques (pilote identifié)
- Opérations sur les périphériques
  - connexion / lecture / écriture / configuration / déconnexion

jean-philippe.babau@univ-brest.fr

UBO

Lôl/O system

- Installations des drivers et des périphériques sous Linux
  - pilote : module noyau contenant des opérations
    - Attribution dôun numéro : le majeur
  - périphérique devName : fichier dans /dev/devName
    - Pour un pilote identifié par son majeur
    - Attribution dôun numéro : le mineur
- Installations des drivers et des périphériques sous VxWorks
  - pilote : librairie dôopérations
  - périphérique : structure de données
- Gestion des opérations sur les périphériques (open, close, read, write, ioctl)
  - API correspondante du driver
    - devOpen, devRelease, devRead, devWrite, devIoctl
  - Lôappel du driver est réalisé par lôOS (**transparente pour lôutilisateur)**
    - recherche du pilote par son numéro majeur
    - appel de la primitive correspondante du pilote
      - Lien open / devOpen, é

jean-philippe.babau@univ-brest.fr

UBO

Primitives du pilote de gestion des périphériques (VxWorks)

- Appelée par l'OS
  - une par appel de l'OS
- Paramètres
  - pointeur vers le descripteur du périphérique
  - paramètres de l'appel de l'OS

```
int  devCreat(DEV * desc, char * name, int  flag) ;

int  devRemove(DEV * desc, char * name) ;

int  devOpen(DEV * desc, char * name, int  flag) ;

int  devClose(DEV * desc, char * name) ;

int devRead (DEV * desc, char * buff, int  nBytes) ;

int devWrite (DEV * desc, char * buff, int  nBytes) ;

int devIOCtl (DEV * desc, int fonction, int  arg) ;
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives de l'OS pour la gestion des pilotes (VxWorks)

- Installation
  - Etablissement du lien primitives IOS / primitives du pilote
  - Mise en place d'un identifiant (numéro du Majeur)

```
drvNumber  = iosDrvInstall ( devCreat, devRemove, devOpen, devClose, devRead, devWrite,
    devIOCtl)

drvNumber  = iosDrvInstall ( devCreat, devRemove,  0, 0, 0, devWrite, devIOCtl)
```

- Désinstallation

```
ret  = iosDrvRemove ( drvNumber,protOpen)
```

  - FALSE : pilote non désinstallé si un périphérique est ouvert
  - TRUE : un périphérique ouvert est fermé et détruit implicitement, le piloté est désinstallé

  - périphérique détruit implicitement
    - Pas d'appel à DevDel, mémoire perdue !

jean-philippe.babau@univ-brest.fr

UBO

Primitives de lôIOS pour la gestion des périphériques (VxWorks)

- Structure de description du périphérique
  - Identifiant
    - Nom
    - Numéro de mineur (Linux)
  - Lien avec le pilote
    - Numéro de Majeur
  - VxWorks : élément dôune liste : DEV_HDR (next, previous, N° de driver, nom)
  - infos spécifiques
    - adresses matérielles, données de configuraiton, é
  - mode R/W (non vérifié par IOS de VxWorks)
  - nombre maximum dôaccès (")

- Ajout dôun périphérique

```
status  = iosDevAdd ( &desc, "name",drvNumber )
```

- Retrait dôun périphérique

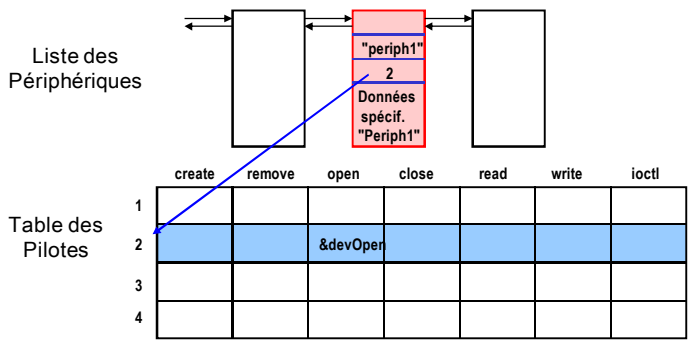
```
status = iosDevDelete( pDevHdr );
```

jean-philippe.babau@univ-brest.fr

UBO

Documentation VxWorks

- Structures de données du système dôE/S nécessaires pour la mise en î uvre dôun pilote
  - Une liste de périphériques, en général gérée dynamiquement
  - Une table des pilotes indexée par le majeur (le majeur désigne un pilote donné)



jean-philippe.babau@univ-brest.fr

UBO

Documentation VxWorks

Å Liste des périphériques

ï Structure de données du descripteur de périphérique

ï typedef struct

```
{
    DEV_HDR devHdr;
    devSpecific autres ;
} DEV;
```

← données spécifiques du périphérique

ï typedef struct

```
{
    DEV_HDR devHdr;
    int flag;
    char data[Taille_MAX];
    é
} DEV;
```

} données spécifiques du périphérique

jean-philippe.babau@univ-brest.fr

UBO

Documentation VxWorks

Å Installation dûn pilote "dev"

pilote_num = iosDrvInstall(&devCreate,0, &devOpen,0,&devRead,&devWrite,&devIoctl);

La fonction définit les routines du pilote pour les 7 fonctions dœE/S (1)

(4) Le système dœE/S renvoie le numéro de pilote Ex : pilote_num=2

Le système dœE/S détermine le prochain Slot libre dans la table des pilotes (2)

Le système dœE/S met à jour les routines du pilote dans la table (3)

Table des Pilotes

	devCreate	devRemove	devOpen	devClose	devRead	devWrite	devIoctl
1							
2	&devCreate	0	&devOpen	0	&devRead	&devWrite	&devIoctl
3							
4							

jean-philippe.babau@univ-brest.fr

UBO

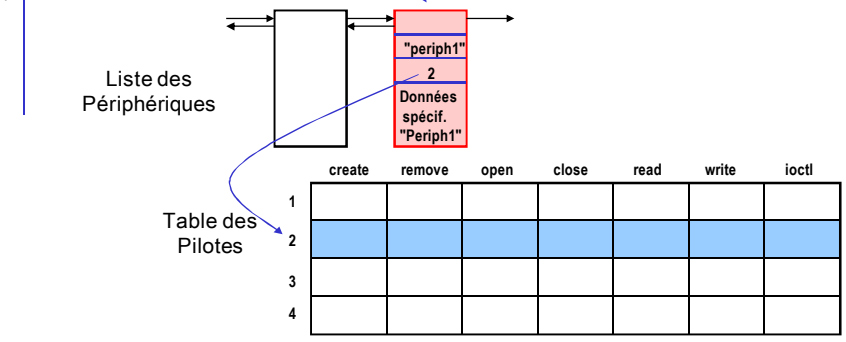
Documentation VxWorks

Ajout de périphériques

```
status = iosDevAdd(&dev0," periph1", 2);
```

renvoie OK, ou ERROR si il existe un périphérique possédant le même nom

Le système de E/S ajoute le descripteur de périphériques à la liste des périphériques



jean-philippe.babau@univ-brest.fr

UBO

Documentation VxWorks

Ouverture d'un périphérique

La primitive travaille sur les données spécifiques du périphérique (4)
Le pilote renvoie un pointeur sur le descripteur de périphérique (5)

Code d'une tâche

```
fd = open(" periph1",O_RDONLY , 0);
```

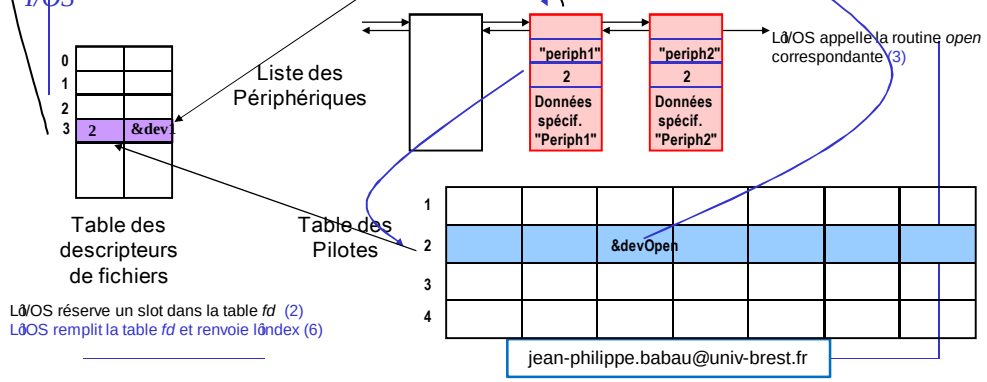
Code du pilote

```
dev1 = devOpen(&dev, " periph1" ,O_RDONLY , 0);
```

I/OS

L'OS trouve le nom dans la liste des périphériques (1)

L'OS appelle la routine *open* correspondante (3)



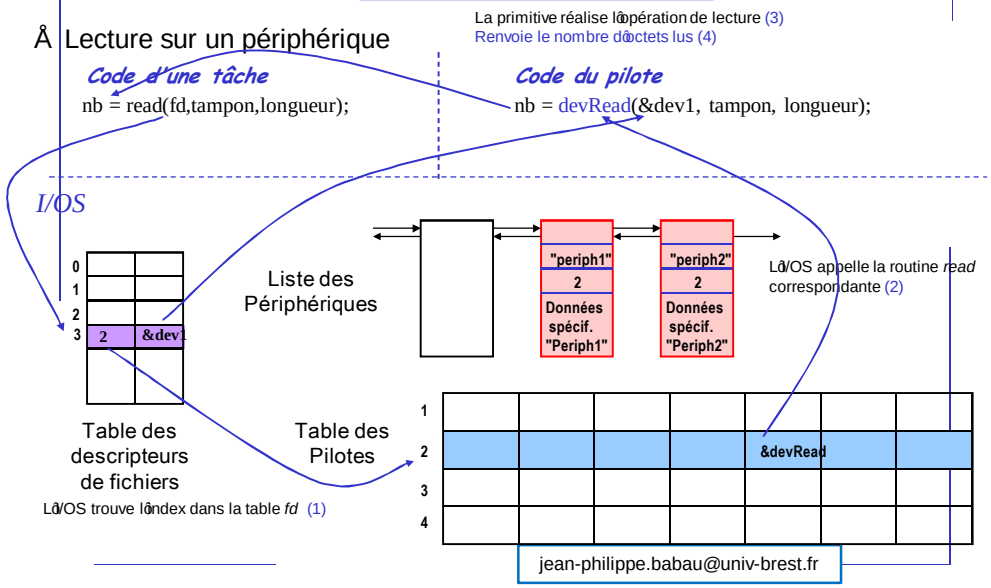
L'OS réserve un slot dans la table *fd* (2)
L'OS remplit la table *fd* et renvoie l'index (6)

jean-philippe.babau@univ-brest.fr

UBO

Documentation VxWorks

Å Lecture sur un périphérique



jean-philippe.babau@univ-brest.fr

UBO

Installation dôun module Linux

Å Module noyau
- ï Macros module_init et module_exit
- ï 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

UBO

Installation dôun module Linux

- 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

UBO

API du pilote pour la gestion des périphériques (Linux)

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

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

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

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

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

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

jean-philippe.babau@univ-brest.fr

UBO

Installation dûun driver sous linux

Â Enregistrement du driver

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

- ï Attribution dynamique dûun numéro majeur si myMajor est à 0
  - Â register_chrdev() renvoie le numéro attribué
- ï Attribution statique dûun numéro majeur sinon
  - Â Utilisation dûun périphérique indépendante de lôinstallation
- ï Liste des numéros majeurs attribués
  - Â Assignés : 0-230; non assignés 231-239; local use : 240-254
  - Â cat /proc/devices

unDriver.h

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

unDriver.c

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

jean-philippe.babau@univ-brest.fr

UBO

Installation dûun driver sous Linux

Â Lien API IOS / API driver

- ï Association primitives IOS / primitives pilote
- ï Structure *file_operations*

```
static struct file_operations fops =
{           open : devOpen,
            read : devRead,
            write : devWrite,
            ioctl : devIoctl,
            release : devRelease
};
```

Â Désenregistrement du driver

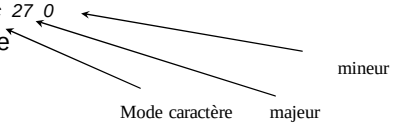
int = unregister_chrdev(unsigned char myMajor, char * drvName);

jean-philippe.babau@univ-brest.fr

UBO

Gestion dûun périphérique sous Linux

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



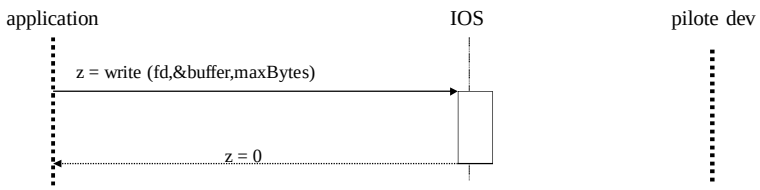

```
#!/bin/sh
driverName="myDriver"
deviceName="myDevice"
modeDevice="a+w"
rm -f /dev/${deviceName}[0-1]
majorNumber=$(awk '$2~/myDriver/ {print $1}' /proc/devices)
mknod /dev/${deviceName}0 c $majorNumber 0
mknod /dev/${deviceName}1 c $majorNumber 1
chmod ${modeDevice} /dev/${deviceName}[0-1]
```

jean-philippe.babau@univ-brest.fr

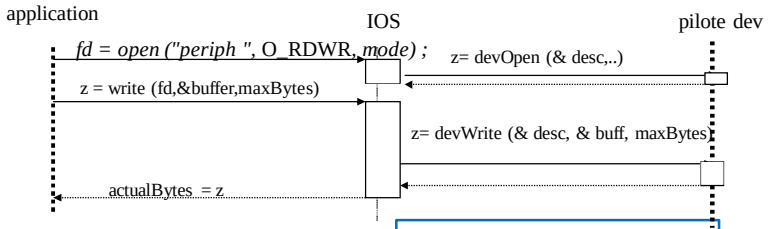
UBO

Comportement de l'IOS

Scénario : écriture sans ouverture préalable ou ouverture incorrecte



Scénario : écriture correcte avec ouverture préalable correcte



jean-philippe.babau@univ-brest.fr

UBO

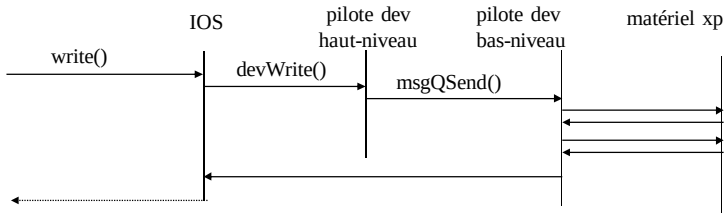
Vérifications

- IOS
  - nom du périphérique
  - connexion
    - pas de read avant open
  - nombre de connexions (non contrôlé par l'IOS de VxWorks)
  - 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

jean-philippe.babau@univ-brest.fr

UBO

Conception du pilote



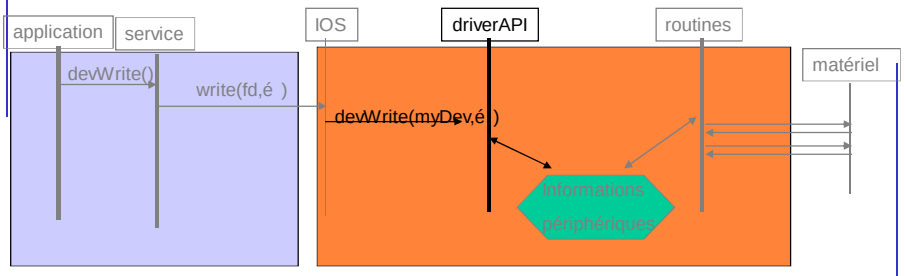
- Partie haut niveau
  - primitives de l'IOS
- Découplage haut et bas niveau
  - boite aux lettres des requêtes
  - conservation de l'ordre d'appel
  - gestion des erreurs

jean-philippe.babau@univ-brest.fr

UBO

Plan

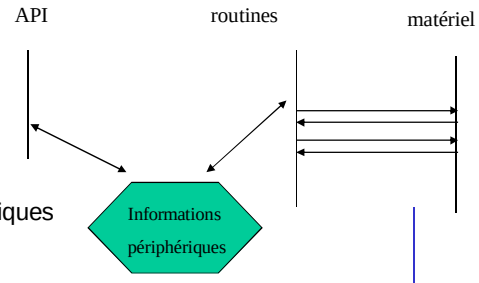
- Pilote intégré
  - Appel de l'IOS
  - Appel du driver
  - Communication bas-niveau



jean-philippe.babau@univ-brest.fr

UBO

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, é
  - Commandes à réaliser
- Le pilote gère et manipule une vue du périphérique
  - Initialisation (adresse, é)
  - Configuration : ioctl
  - Connexion : open/close ou creat/remove
  - Lecture / écriture : read / write

jean-philippe.babau@univ-brest.fr

UBO

Gestion des données périphériques

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

unDriver.h

```
#define MAXDEVICES 16

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

jean-philippe.babau@univ-brest.fr

UBO

Interface du pilote VxWorks

- Primitives de gestion du pilote et déclaration de types

```
#ifndef PILOTE_H
#define PILOTE_H

#include <iosLib.h>             /* librairie pour les pilotes */

typedef struct                  /* structure d'échange des données périphériques : cast du buffer */
{       int valeur ;
} T_data ;

int DrvInstall();               /* primitive d'installation du pilote */
int DrvRemove();                /* primitive de désinstallation du pilote */

int DevAdd( char * name);       /* primitive d'ajout d'un périphérique */

int DevDel(char * name);        /* primitive de suppression d'un périphérique */

int  DrvConfig();               /* primitive de configuration */
int  InfoDrv();                 /* primitive de test */

#endif
```

jean-philippe.babau@univ-brest.fr

UBO

Réalisation du pilote VxWorks

Pilote.c

```
#include ñPilote.hò
#include <stdlib.h>

typedef struct          /* structure spécifique pour un périphérique */
{    int numero ;
} specific ;

typedef struct          /* structure standard  VxWoks pour un périphérique */
{    DEV_HDR donnees;
     specific autres;
} DEV;

/* Déclarations et implémentation des 7 primitives devCreat, devRemove, devOpen,
     devClose, devRead, devWrite, devIOCtl */

/* Implémentation des primitives déclarées dans Pilote.h */
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives d'installation VxWorks

```
int drvNumber = -1; int nbrePeriph = 0;

int DrvInstall()
{
     if (drvNumber==-1)
     {
     /* exemple d'installation de pilote */
     drvNumber  = iosDrvInstall ( &devCreat, &devRemove, &devOpen, &devClose, &devRead,
     &devWrite, &devIOCtl) ;

     /* exemple d'installation de pilote */
     drvNumber  = iosDrvInstall ( NULL, NULL, &devOpen, &devClose,      NULL, &devWrite,
     &devIOCtl) ;
     }

     return drvNumber;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives d'installation VxWorks

```
int DevAdd(char * name)
{
DEV * desc = (DEV *)malloc(sizeof(DEV));

      if (drvNumber != -1)
      {

      (desc->autres).numero = nbrePeriph ++ ;

      iosDevAdd ((DEV_HDR *)desc, name, drvNumber);

      return nbrePeriph ;
      }
else
      return -1;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives de désinstallation VxWorks

```
int DevDel(char * name)
{
      DEV_HDR * pDevHdr;        /* structure pour le périphérique */
      char* suite [1];

      pDevHdr = iosDevFind(name,suite);/* recherche du périphérique */
      if ((pDevHdr!= NULL) &&  (*suite[0]==ô\0ô))
      {
            iosDevDelete( pDevHdr );
            free(pDevHdr);
      }
      return 0;
}

int DrvRemove()
{     int ret;                /* numéro du pilote */
      if (drvNumber != -1)
      {     ret  = iosDrvRemove( drvNumber,FALSE) ;
            drvNumber = -1 ;}
      return ret ;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives liés au périphérique VxWorks

```
int  devOpen(DEV * desc, char * remainder, int  mode)
{
    if (*remainder!=ô0ô)
    {
        return ERROR;
    }
    else
    {
        return ((int) desc) ;
    }
}

int  devClose(DEV * desc, char * name)
{
    return 0;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Primitives liés au périphérique VxWorks

```
int devWrite (DEV * desc, char * buff, int  nBytes)
{
    /* pas de printf dans un driver !! */

    printf("periph %d : %s \n",    (desc->autres).numero , buff);

    return 0;
}

int devIOCtl (DEV * desc, int fonction, int  arg)
{
    if (fonction= =1)
    {
        (desc->autres). numero = - (desc->autres). numero ;
    }
    return 0;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Appel du pilote (VxWorks)

```
#include " Pilote.h"

int main(void)
{
int numPilote;
int fd1;
int fd2;

numPilote = DrvInstall();

DevAdd( "periph1");
DevAdd( "periph2");

fd1=open("periph1",O_RDWR,0);
fd2=open("periph2",O_RDWR,0);

write(fd1,"Hello",6) ;
write(fd2," world",7) ;

ioctl(fd1,0,0);
ioctl(fd2,1,0);

write(fd2," world",7) ;

close (fd2);
close (fd1);

DrvRemove();

return (0);}
```

jean-philippe.babau@univ-brest.fr

UBO

Initialisation des données via l'installation (Linux)

```
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;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Association informations des périphériques ï file system (Linux)

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

```
static ssize_t devOpen(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; }
}
```

jean-philippe.babau@univ-brest.fr

UBO

Désinstallation et release (Linux)

ï 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 devRelease(struct inode *inode, struct file* file)
{
        return 0;
}
```

jean-philippe.babau@univ-brest.fr

UBO

Opération de transfert : écriture (Linux)

- 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 devWrite(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

UBO

Opération de transfert : lecture (Linux)

- 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 devRead(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

UBO

Primitive ioctl (Linux)

- Primitive spécifique
  - Utlisation normale en configuration

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

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

jean-philippe.babau@univ-brest.fr

UBO

Primitive ioctl (Linux)

- 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  'dô

#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

UBO

Primitive ioctl (Linux)

- 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 devIoctl(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

UBO

Appel du pilote (Linux)

```
#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

UBO

Pilotes fournis par l'OS

- Pilote existants
  - Périphériques intégrés
  - pilote terminal
  - *pilote de fichiers*
  - *pilote de pipe*

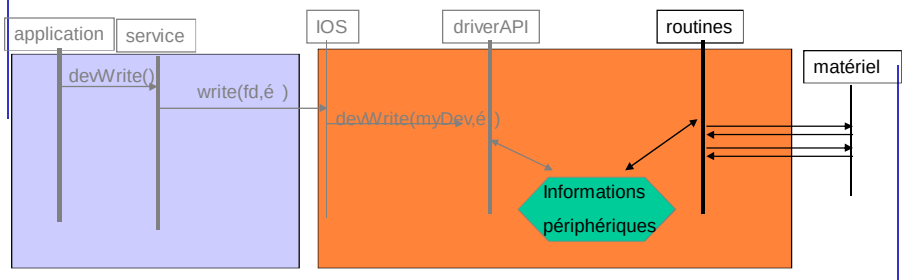
- Pilote terminal
  - ttyS0
  - paramétrisation via ioctl
  - fonctions
    - FIOSETOPTIONS, OPT_LINE (arrêt sur NEWLINE)
    - FIOBAUDRATE, vitesse
    - FIOFLUSH, 0
    - FIOCANCEL (arrêt d'un read/write), 0

jean-philippe.babau@univ-brest.fr

UBO

Plan

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



jean-philippe.babau@univ-brest.fr

UBO

OS monolithiques

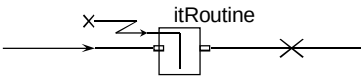
- L'OS peut être vu comme une application qui offre des services
  - Driver : des services **de** l'OS
  - Pas de concurrence au sein du driver
- Séparation application / noyau
  - Gestion de la mémoire
    - Application : adressage virtuel
    - Noyau : adressage réel
  - Communication de données
    - Mécanismes spécifiques de copie
    - Linux : *copy_to_user*, *copy_from_user*
  - Protection du système
  - Primitives noyaux pour le pilote
- Gestion du bas niveau
  - Gestion des IT
    - Windows : ISR -> DPC
  - Gestion DMA

jean-philippe.babau@univ-brest.fr

UBO

L'interruption sous Linux

Création



Destruction

C/POSIX

```
#include <linux/interrupt.h>
static irqreturn_t itRoutine(int irq, void* devID)
{ é
return IRQ_HANDLED;}
é
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

UBO

Le sémaphore Linux 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
```

jean-philippe.babau@univ-brest.fr

UBO

Spinlock (Linux)

- 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
```

jean-philippe.babau@univ-brest.fr

UBO

Couplage avec le matériel (Linux)

- 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

UBO

Primitives spécifiques mode noyau Linux

- 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

UBO

Plan

- Besoins applicatifs en terme de communication
  - Configurer les périphériques
  - Echanger des informations (en entrée ou en sortie) avec lôenvironnement
  - *Communication dôinformations (cf. protocoles réseau)*
- Implémentation des pilotes
  - Mise en î uvre dans les OS monolithiques (Linux)
  - Pilotes intégrés
    - Périphérique vu comme un fichier
  - Couplage matériel / logiciel
- Architecture des logiciels de communication
  - Allocation
  - Architecture en couches
  - Gestion de services
    - Politique dôaccès et de partage des services

jean-philippe.babau@univ-brest.fr

UBO

Primitives dôinitialisation / terminaison

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

jean-philippe.babau@univ-brest.fr

UBO

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, é
  - Attente infinie de données
  - Données invalides

jean-philippe.babau@univ-brest.fr

UBO

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)

jean-philippe.babau@univ-brest.fr

UBO

Architecture en couches

- Driver de driver é
- 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

jean-philippe.babau@univ-brest.fr

UBO

Exemple

```
#define DevicePortName COM1

int installed = 0;

int deviceOpen()
{
    if !(installed)
    {
        installed = open(" DevicePortName ",O_WRONLY);
        /*ouverture dûne liaison série en écriture */
        if (installed < 0)
        {       /* mode trace : printf(« erreur connexion périphérique\n"); */
                installed = 0;
                return (-1);
        }
        else
        {       /* mode trace : printf(« connexion périphérique OK\n"); */
                return (1);
        }
    else
    {   /* mode trace : printf(«périphérique deja connecte\n"); */
        return (-2);
    }
}
```

jean-philippe.babau@univ-brest.fr

UBO

Exemple

```
int deviceClose( int devDesc)
{
    if !(installed)
    {
        /* mode trace : printf(«périphérique non connecte\n"); */
        return (-1);
     }
    else
    {   /* mode trace : printf(« connexion périphérique OK\n"); */
        close (installed);
        installed = 0;
        return (1);
    }
}
```

- Connexion
  - Connexion au port série, pas au périphérique é

jean-philippe.babau@univ-brest.fr

UBO

Exemple

```
int deviceWrite( char cmde, char argument)
{
    char bufferE[8];
    int crc;

    if !(installed)
    {
        /* mode trace : printf(«périphérique non connecte\n"); */
        return (-1);
     }
    else
    {   /* mode trace : printf(« connexion périphérique OK\n"); */

        bufferE [0] = 1 ; bufferE[1] = 0 ; bufferE[2] = 2 ;
        bufferE [3] = cmde ; bufferE[4] = argument ;
        crc = CRC(bufferE,5);
        bufferE [5] = char(crc) ; bufferE[6] = char(crc)>>8 ;
        bufferE[7] = 13 ;

        write (installed,bufferE,8);          /* envoi trame de commande périphérique */
    }
}
```

- Protocole sans acquittement

jean-philippe.babau@univ-brest.fr

UBO

La couche service

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

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

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

jean-philippe.babau@univ-brest.fr

UBO

La couche service

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

jean-philippe.babau@univ-brest.fr

UBO

La couche service

- Fonctions avancées
  - accès multiples
    - creat ( ); creat ( );
  - 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

UBO

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

UBO

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

UBO

Périphérique virtuel

- Définition de périphériques virtuels
  - Élément de lôIHM (souris, clavier, écran)
  - VirtualMouse de Windows
- Pilote de périphériques virtuels
  - Livré avec lôOS
  - 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

UBO

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

UBO

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

UBO

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 dôune 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

UBO

Références utiles et utilisées pour ce cours

- *broux.ftp-developpez.com/articles/c/driver-c-linux.pdf*
- J. Corbet, A. Rubini, G. Kroah-Hartman ñLINUX DEVICE DRIVERSò OôReilly Media, ISBN-13 : 978-0-596-00590-0, 2005

jean-philippe.babau@univ-brest.fr