

Différentes étapes de la mise en place du module

Jérôme Ermont

IRIT - Toulouse INP/ENSEEIH

2023 – 2024

En aperçu

- ▶ L'objectif est de construire un module Linux qui réalise la communication entre 2 stations connectées par un câble parallèle
- ▶ Mise en œuvre du protocole **crynwr**
- ▶ Le TP suit les étapes :
 1. Écriture et compilation d'un module (qui affiche un message au chargement et un message au déchargement). Interfaçage avec le sous-système de gestion des ports parallèle.
 2. Émission et réception d'un octet.
 3. Émission et réception d'un tableau d'octets (construit statiquement). La réception se fera en dehors du gestionnaire d'interruption.
 4. Interfaçage avec le sous-système réseau.
 5. Construction et émission d'une trame. Réception et extraction du paquet contenu.

Etape 1 : compiler un module dans le noyau Linux

- ▶ Le document de référence :
https://moodle-n7.inp-toulouse.fr/pluginfile.php/147448/mod_folder/content/0/compiler-linux.pdf?forcedownload=1.
- ▶ Compiler un module : `make`
- ▶ Insertion du module : `insmod l2p.ko`
- ▶ Visualisation des modules lancés : `lsmod`
- ▶ Désinstallation d'un module : `rmmod l2p`

Etape 1 : Mise en place de l'interface avec le port parallèle

- ▶ Documentation sur le port parallèle :
- ▶ Définition du pilote utilisant le port parallèle :

```
static struct parport_driver l2p_ppdrv= {  
    .name = "l2p",  
    .attach = l2p_attach,  
    .detach = l2p_detach  
};
```

- ▶ La fonction l2p_attach est lancée à l'enregistrement :

```
void l2p_attach(struct parport *pport) {  
    ...  
}
```

- ▶ l2p_detach est exécutée lors du désenregistrement :

```
void l2p_detach(struct parport *pport) {  
    ...  
}
```

Etape 1 : Mise en place de l'interface avec le port parallèle

- ▶ Enregistrement du pilote (voir page 96) :

```
if (parport_register_driver(&l2p_ppdrv) != 0)
{
    printk("l2p: parport_register_driver
failed\n");
    return -EIO;
}
```

- ▶ Désenregistrement du pilote (voir page 96) :

```
parport_unregister_driver(&l2p_ppdrv);
```

- ▶ Après l'enregistrement du pilote `l2p_attach` est automatiquement appelé
 - ▶ Dans cette fonction, notre client peut s'enregistrer auprès du port parallèle

Etape 1 : Mise en place de l'interface avec le port parallèle

- Enregistrement du client du port parallèle :

```
struct pardevice *ppdev= NULL;  
ppdev= parport_register_device(pport, "l2p",  
    l2p_preempt, l2p_wakeup, l2p_irq, 0, &  
    l2p_device)
```

- La variable l2p_device permet de stocker les données de notre protocole :

```
struct l2p_dev {  
    struct pardevice *ppdev;  
};  
struct l2p_dev l2p_device;
```

Il sera alors possible de stocké dans cette structure le contenu de la variable ppdev :

```
l2p_device.ppdev= ppdev;
```

Etape 1 : Mise en place de l'interface avec le port parallèle

Attention !

Il n'y a plus de protection contre les erreurs d'accès mémoire illégaux dans le noyau. Pensez à ce qui peut se passer si `ppdev == NULL`.

- ▶ Désenregistrement du client du port parallèle :

```
parport_unregister_device(ppdev);
```

- ▶ Les fonctions `l2p_preempt`, `l2p_wakeup` et `l2p_irq` ont les interfaces suivantes (voir page 96) :

```
int l2p_preempt(void *data);  
void l2p_wakeup(void *data);  
void l2p_irq(void *data);
```

- ▶ Le champs `data` contiendra les données de `l2p_device`
- ▶ Seul `l2p_irq` sera exécuté à la réception d'une interruption. Les 2 autres seront vides.

Etape 1 : Mise en place de l'interface avec le port parallèle

- Pour pouvoir lire et écrire sur le port parallèle, il faut le réclamer (p96) :

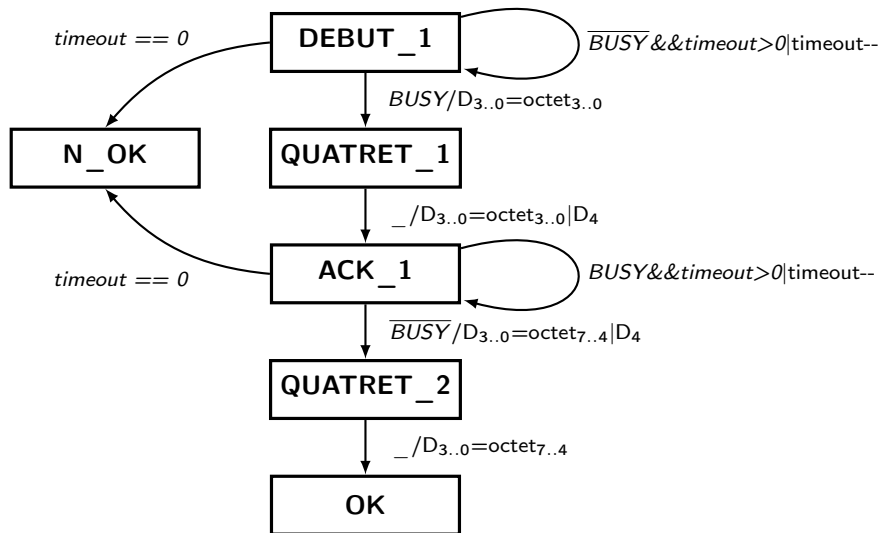
```
if (parport_claim(ppdev)) {  
    printk("l2p: parport_claim failed \n");  
    return -1;  
}
```

- Si le port est obtenu, il est possible d'émettre et de recevoir un caractère.
- Libération du port parallèle :

```
parport_release(ppdev);
```


Etape 2 : Envoyer un caractère

- Automate de fonctionnement :



Etape 2 : Envoyer un caractère

- ▶ La fonction d'émission :

```
int emettre_octet(unsigned char octet, struct
    parport *pport) {
    ....
}
```

- ▶ Lecture sur le port parallèle :

```
unsigned char status;
status= pport->ops->read_status(pport);
```

- ▶ Ecriture sur le port parallèle :

```
unsigned char quatret;
pport->ops->write_data(pport, quatret);
```

- ▶ Les états de l'automate :

```
enum sender_states_t {DEBUT_1, QUATRET_1, ACK_1,
    QUATRET_2, OK, N_OK};
enum sender_states_t sender_state= DEBUT_1;
```

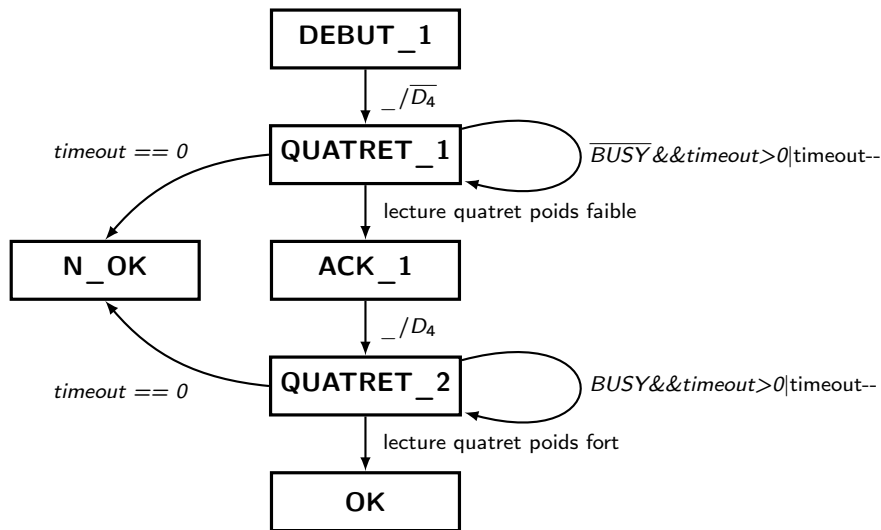
Etape 2 : Envoyer un caractère

► Exécution de l'automate :

```
switch (sender_state) {
    case DEBUT_1:
        status= pport->ops->read_status(pport);
        if (ISBUSY(status)) {
            quatret= ...;
            pport->ops->write_data(quatret);
            sender_state= QUATRET_1;
        }
        if (timeout>0 && !(ISBUSY(status))) {
            timeout--;
            udelay(TIMEOUT_DELAY);
        } else if (timeout == 0) {
            sender_state= N_OK;
        }
        break;
    case QUATRET_1:
        ...
}
```

Etape 2 : Recevoir un caractère

- Automate de fonctionnement :




Etape 2 : Recevoir un caractère

- ▶ La fonction de réception :

```
int recevoir_octet(unsigned char /*inout*/ *octet,  
    struct parport *pport) {  
    ....  
}
```

- ▶ Attention : c'est un choix possible mais non obligatoire, cela permet ici d'indiquer comment s'est terminée la fonction.
- ▶ Dans un 1er temps, les fonctions d'émission et de réception se lanceront à l'enregistrement du pilote auprès du port parallèle, dès que le port parallèle sera réclamé et obtenu.
- ▶ Une station émetteur
- ▶ Une station récepteur

Etape 2 : Utilisation des interruptions pour la réception

- ▶ Objectif : commencer la réception du caractère dès qu'un signal est émis sur le port parallèle
- ▶ Handler d'interruption : `l2p_irq`
- ▶ Générer une interruption : envoi D3 
- ▶ Activer les interruptions :

```
pport->ops->enable_irq();
```

- ▶ Désactiver les interruptions :

```
pport->ops->disable_irq();
```

- ▶ En réception, bit ACK == 1
 - ▶ A vérifier pour prévenir de toute interruption parasite

Etape 3 : Emission et réception d'un tableau de caractères

- ▶ La taille du tableau est fixe et connue de l'émission et de la réception
 - ▶ Par exemple : `#define TAB_SIZE 5`
- ▶ Ecrire 2 fonctions :

```
int emettre_tab(char tab[], int tab_size, struct  
    parport *pport);  
int recevoir_tab(char tab[], int tab_size, struct  
    parport *pport);
```

qui appellent `tab_size` fois `emettre_octet` OU `recevoir_octet`

- ▶ On peut choisir de renvoyer le nombre d'octets lus ou écrits.
- ▶ Tester ces fonctions en lançant l'émission dans la fonction `l2p_attach` avec un tableau qui contient une chaîne de caractères (par exemple : `tab[TAB_SIZE] = "toto"`)

Etape 3 : Différer la réception dans une tâche

- ▶ Objectif : Réduire le temps de traitement dans le handler d'IT
- ▶ Comment ?
 - ▶ Exécuter le code qui prend du temps dans une tâche spécifique
 - ▶ Utilisation des Workqueues (voir Time Delays and Deferred Work, p205)
- ▶ Définition de la tâche :

```
struct work_struct reception;
```

- ▶ Initialisation et association à la fonction de réception :

```
INIT_WORK(&reception, recevoir_trame);
```

- ▶ La fonction de réception associé doit avoir le format suivant :

```
void recevoir_trame(struct work_struct *work);
```

- ▶ Lancement de la tâche :

```
schedule_work(&reception);
```


Etape 4 : Interfaçage avec la pile IP

- ▶ La structure `net_device` (Network Drivers, p 502) permet de référencer notre interface de communication avec IP
- ▶ Allocation d'une `net_device` :

```
struct net_device dev= alloc_netdev(  
    sizeof(struct l2p_dev), // ou 0  
    "l2p%d", // nom de l'interface  
    NET_NAME_UNKNOWN,  
    l2p_devinit // configuration de l'interface  
);
```

- ▶ `sizeof(...)` : la taille des données privées, si besoin
- ▶ `"l2p%d"` : le nom de l'interface tel qu'il apparaît dans `ifconfig` ou `ip show`; `%d` sera remplacé le numéro de l'interface, par exemple `l2p0`, `l2p1`, ...
- ▶ la fonction `l2p_devinit` (à rajouter) permet de configurer l'interface réseau. La signature de cette fonction est :

```
void l2p_devinit(struct net_device *dev);
```

Etape 4 : Interfaçage avec la pile IP

Contenu de la fonction `l2p_devinit`

- ▶ Objectif de la fonction : initialiser les différents champs de la structure :
 - ▶ Taille du buffer d'échange, MTU, Fonctions `open` et `stop` exécutées lorsque l'interface passe à up et à down, Fonction `hard_xmit` appelée par IP pour émettre une trame
 - ▶ et plein d'autres champs
- ▶ Pour initialiser les champs à la façon d'une interface Ethernet, il suffit d'appeler la fonction `ether_setup` :

```
ether_setup(dev);
```

- ▶ Configuration spécifique à notre interface :
 - ▶ L'interface est point à point et ne dispose pas d'ARP :

```
dev->flags= IFF_POINTOPOINT|IFF_NOARP;
```

- ▶ Taille du buffer d'échange :

```
dev->tx_queue_len= 10;
```

- ▶ MTU :

```
dev->mtu= 1500;
```

Etape 4 : Interfaçage avec la pile IP

Contenu de la fonction `l2p_devinit`

- Configuration spécifique à notre interface :

- Adresse MAC de l'interface :

- ```
memset(dev->dev_addr, 0xfc, ETH_ALEN);
```

- Initialisation des fonctions liées à l'interface :

- ```
dev->netdev_ops = &l2p_netdev_ops;
```

- Initialisation des headers :

- ```
dev->header_ops = &l2p_header_ops;
```

## Etape 4 : Interfaçage avec la pile IP

- ▶ `l2p_netdev_ops` est définie par :

```
const struct net_device_ops l2p_netdev_ops = {
 .ndo_open = l2p_open,
 .ndo_stop = l2p_close,
 .ndo_start_xmit = emettre_trame,
 .ndo_do_ioctl = NULL,
};
```

- ▶ `l2p_open` se lance lorsque l'interface passe à up :

```
int l2p_open(struct net_device *dev) {
 ...
 return 0;
}
```

- ▶ `l2p_close` se lance lorsque l'interface passe à down :

```
int l2p_close(struct net_device *dev){
 ...
 return 0;
}
```

## Etape 4 : Interfaçage avec la pile IP

- ▶ `ndo_start_xmit` lance la transmission d'une trame. Elle est couplé à notre fonction d'émission de trame :

```
int emettre_trame(struct sk_buff *skb, struct
 net_device *dev) {
 ...
 return 0;
}
```

- ▶ La trame à émettre est contenue dans la structure `struct sk_buff *skb`.
- ▶ Cette fonction fera appel à `emettre_octet` que nous avons défini.

## Etape 4 : Interfaçage avec la pile IP

- ▶ l2p\_header\_ops est définie par :

```
const struct header_ops l2p_header_ops = {
 .create= NULL,
 .cache= NULL,
};
```