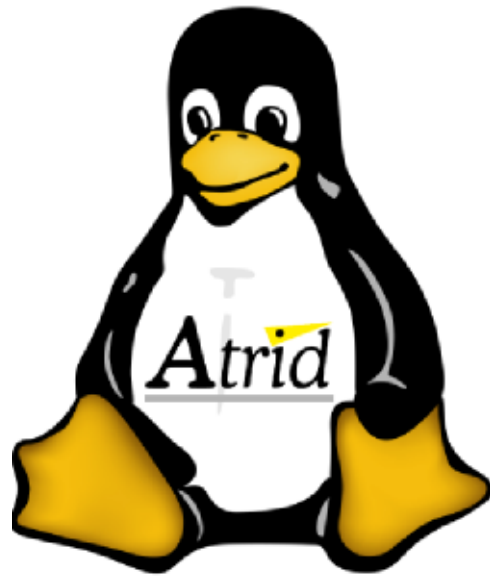


PRÉSENTATION DE RT-LINUX



ATRID

PRÉSENTATION DE RT-LINUX

par ATRID

Copyright © 1999-2000 par ATRID Systèmes

Ce document peut être librement lu, stocké, reproduit, diffusé, traduit et cité par tous moyens et sur tous supports aux conditions suivantes:

- Tout lecteur ou utilisateur de ce document reconnaît avoir pris connaissance de ce qu'aucune garantie n'est donnée quant à son contenu, à tous points de vue, notamment véracité, précision et adéquation pour toute utilisation ;
- il n'est procédé à aucune modification autre que cosmétique, changement de format de représentation, traduction, correction d'une erreur de syntaxe évidente, ou en accord avec les clauses ci-dessous ;
- le nom, le logo et les coordonnées de l'auteur devront être préservés sur toutes les versions dérivées du document à tous les endroits où ils apparaissent dans l'original, les noms et logos d'autres contributeurs ne pourront pas apparaître dans une taille supérieure à celle des auteurs précédents, des commentaires ou additions peuvent être insérés à condition d'apparaître clairement comme tels ;
- les traductions ou fragments doivent faire clairement référence à une copie originale complète, si possible à une copie facilement accessible ;
- les traductions et les commentaires ou ajouts insérés doivent être datés et leur(s) auteur(s) doi(ven)t être identifiable(s) (éventuellement au travers d'un alias) ;
- cette licence est préservée et s'applique à l'ensemble du document et des modifications et ajouts éventuels (sauf en cas de citation courte), quelqu'en soit le format de représentation ;
- quel que soit le mode de stockage, reproduction ou diffusion, toute version imprimée doit contenir une référence à une version numérique librement accessible au moment de la première diffusion de la version imprimée, toute personne ayant accès à une version numérisée de ce document doit pouvoir en faire une copie numérisée dans un format directement utilisable et si possible éditable, suivant les standards publics, et publiquement documentés en usage ;

La transmission de ce document à un tiers se fait avec transmission de cette licence, sans modification, et en particulier sans addition de clause ou contrainte nouvelle, explicite ou implicite, liée ou non à cette transmission. En particulier, en cas d'inclusion dans une base de données ou une collection, le propriétaire ou l'exploitant de la base ou de la collection s'interdit tout droit de regard lié à ce stockage et concernant l'utilisation qui pourrait être faite du document après extraction de la base ou de la collection, seul ou en relation avec d'autres documents.

Toute incompatibilité des clauses ci-dessus avec des dispositions ou contraintes légales, contractuelles ou judiciaires implique une limitation correspondante : droit de lecture, utilisation ou redistribution verbatim ou modifiée du document.

Adapté de la licence Licence LLDD v1, octobre 1997, Libre reproduction © Copyright Bernard Lang [F1450324322014] URL :

<http://pauillac.inria.fr/~lang/licence/lldd.html>

Historique des version

Version 1.0 du 10/04/2000

RT-Linux Version 2.3

Version 1.1 du 11/12/2000

Conversion en SGML/Docbook

Table des matières

1. Introduction.....	4
1.1. Présentation	4
1.2. Installation.....	4
2. Bibliothèque de fonctions	6
2.1. Fonctions de base	6
2.1.1. Gestion des interruptions	6
2.1.2. Horloge	7
2.2. Gestion des tâches	8
2.3. Gestion des MUTEX.....	10
2.4. Gestion des pilotes	10
2.5. Gestion des FIFOs.....	11
3. Mise en oeuvre.....	12
3.1. Déclenchement périodique	12
3.2. Producteur-consommateur	12
3.3. Pilote de périphérique	14
A. Bibliographie	18

Chapitre 1. Introduction

1.1. Présentation

RT-Linux est une extension de Linux apportant une dimension temps réel au noyau standard. Cette extension a été développée par Victor Yodaiken et Michael Barabanov du "Department of Computer Science of the Institute for Mining and Technology of New Mexico".

Le présent document présente la version 2 de RT-Linux, cette version utilisant une interface de programmation POSIX.

Le principe de cette extension est d'ajouter un petit noyau temps réel gérant l'ordonnancement des tâches temps réel et les interruptions matérielles à la place des outils standards du noyau Linux.

Ce principe permet de s'appuyer sur l'environnement Linux pour toutes les tâches non critiques et donc de bénéficier des couches réseau, du graphisme X-Window et des applications de stockage et de mise en forme des données.

Lorsqu'il n'y a aucune tâche temps réel à exécuter, le système RT donne la main à Linux pour l'exécution des tâches non critiques (Linux est la tâche de plus basse priorité du noyau temps réel).

Les interruptions sont gérées par le noyau RT et transmises au noyau Linux en fonction de son besoin et de son état. Les tâches temps réel s'exécutent en mode noyau et peuvent donc accéder au matériel sans problème (interruptions, registre d'E/S, DMA, ...). Par contre la programmation en mode noyau interdit l'allocation dynamique de mémoire et ne permet pas d'accéder aux données dans l'espace utilisateur des processus ou aux appels systèmes Linux directement.

RT-Linux propose un moyen de communication entre les tâches temps réel et les processus utilisateurs basé sur les FIFOs. Ces fichiers spéciaux sont vus et utilisés, coté processus, à l'aide des appels systèmes classiques de gestion des fichiers sous Linux. Coté RT-Linux, une bibliothèque de fonctions est proposée pour permettre l'envoi et la réception de données par ce biais.

RT-Linux version 2 est structuré en un module principal et plusieurs modules optionnels :

- le module principal implémente la gestion des interruptions et le dialogue avec le noyau linux
- le module d'ordonnancement (`rtl_sched`) implémente une API de gestion des tâches à la mode POSIX ainsi qu'une bibliothèque de compatibilité avec la version 1.
- le module de gestion d'horloge (`rtl_time`) contrôle les horloges systèmes et offre un jeu de fonctions pour les utiliser
- le module d'E/S (`rtl_posixio`) fournit un interface de type POSIX pour les drivers
- le module de gestion des FIFOs (`rtl_fifo`) permet la communication avec les processus utilisateur
- un module de gestion de sémaphores
- un module de gestion de la mémoire partagée

1.2. Installation

L'installation s'effectue soit sur un noyau existant en appliquant des "patches" soit directement à partir d'un noyau préparé. Dans les deux cas, il suffit de générer un noyau Linux avec les sous ensembles souhaités. Pour un système de développement, il est fortement conseillé de valider l'utilisation des modules chargeables de manière à ne pas être obligé de régénérer un noyau à chaque modification du code d'une tâche temps réel.

Après compilation du noyau et des modules, les tâches temps réel sont installées dans le noyau grâce aux commandes de gestion des modules (**insmod**, **lsmod** et **rmmmod**).

Chapitre 2. Bibliothèque de fonctions

Le développement sous RT-Linux s'effectue en écrivant des modules pour le noyau Linux. Ces modules seront gérés par l'ordonnanceur temps réel et doivent être aussi concis que possible. Une des grandes règles d'écriture sous RT-Linux est de ne mettre dans ces tâches que ce qui concerne le temps réel.

RT-Linux version 2 tente de proposer une API conforme à la norme POSIX 1003.1 dans son application aux petits systèmes temps réel minimaux proches du matériel. Il introduit des extensions pour la gestion du SMP.

Les fonctions de RT-Linux disponibles pour l'écriture des tâches temps réel sont présentées en les classant en fonction des modules qui les implémentent.

2.1. Fonctions de base

Les fonctions de base intègrent la gestion des interruptions matérielles et logicielles qui permettent de communiquer avec le noyau Linux standard.

2.1.1. Gestion des interruptions

```
int rtl_request_irq (unsigned int irq, unsigned int (*handler)(unsigned int irq, struct pt_regs *regs))
int rtl_free_irq(unsigned int irq)
```

Ces fonctions permettent d'attacher et de détacher une fonction de gestion d'interruption à un niveau d'interruption IRQ. Elles sont généralement appelées à partir des fonctions `init_module` et `cleanup_module` respectivement.

```
int rtl_hard_enable_irq(unsigned int irq)
int rtl_hard_disable_irq(unsigned int irq)
```

Ces fonctions permettent de valider ou d'inhiber une interruption matérielle.

```
int rtl_get_soft_irq(void (*handler)(int, void *, struct pt_regs *), const char * devname)
void rtl_free_soft_irq(unsigned int irq)
void rtl_global_pend_irq(int irq)
```

La fonction `rtl_get_soft_irq` permet d'allouer un vecteur d'interruption logiciel associé à une fonction pouvant travailler dans le noyau de Linux, c'est à dire utiliser toutes les fonctions utilisables par un gestionnaire d'interruption sous Linux. Elle retourne le vecteur alloué qui peut être libéré par `rtl_free_soft_irq`. Le fichier `devname` est créé dans `/proc/interrupts` pour permettre un accès à ce vecteur d'interruption. La fonction `rtl_global_pend_irq` permet de déclencher une interruption logicielle.

```
void rtl_stop_interrupts()
void rtl_allow_interrupts()
```

Ces fonctions permettent de bloquer et d'autoriser les interruptions sur le processeur courant.

```
void rtl_no_interrupts(rtl_irqstate_t state)
```

```
void rtl_restore_interrupts(rtl_irqstate_t state)
```

Ces fonctions permettent de bloquer et d'autoriser les interruptions sur le processeur courant en sauvegardant l'état courant.

Une utilisation courante de ces fonctions est donnée dans l'exemple suivant pour l'installation d'une fonction de gestion à l'initialisation d'un module :

```
int init_module(void)
{
    rtl_irqstate status;
    rtl_no_interrupts(status);
    res = rtl_request_irq(MY_IRQ, driver_isr);
    if (res < 0)
    {
        printk("error rtl_request_irq ...");
    }
    else
    {
        // initialisation du matériel
        rtl_hard_enable_irq(MY_IRQ);
    }
    rtl_restore_interupts(status);
}
```

2.1.2. Horloge

RT-Linux offre une interface de gestion de l'horloge basée sur une structure et quelques fonctions de manipulation de cette structure.

```
extern clockid_t rtl_getbestclock (unsigned int cpu)
```

Cette fonction retourne la "meilleure" horloge disponible pour le CPU. C'est le point d'entrée pour la manipulation de l'horloge car les fonctions sont disponibles comme membres de cette structure :

```
struct rtl_clock {
    int (*init) (struct rtl_clock *);
    void (*uninit) (struct rtl_clock *);
    hrtime_t (*gethrtime)(struct rtl_clock *);
    int (*sethrtime)(struct rtl_clock *, hrtime_t t);
    int (*settimer)(struct rtl_clock *, hrtime_t interval);
    int (*settimermode)(struct rtl_clock *, int mode);
    clock_irq_handler_t handler;
    int mode;
    hrtime_t resolution;
}
```

```
    hrtime_t value; /* only makes sense for periodic clocks */
    hrtime_t delta;
    struct rtl_clock_arch arch;
};
```

La structure est définie en tant que type `clockid_t`.

```
extern int rtl_setclockhandler (clockid_t h, clock_irq_handler_t fn)
extern int rtl_unsetclockhandler (clockid_t h)
```

Ces fonctions permettent de mettre en place une fonction de gestion de l'interruption horloge pour une tâche donnée.

```
int rtl_setclockmode(clockid_t clock, int mode, hrtime_t mode_param);
```

Cette fonction permet de spécifier le mode de fonctionnement de l'horloge. Le mode peut être :

- `RTL_CLOCK_MODE_ONESHOT` : l'horloge fonctionne une seule fois (`mode_param` est ignoré)
- `RTL_CLOCK_MODE_PERIODIC` : l'horloge est périodique et `mode_param` donne la période en nanoseconde

Le module de gestion de l'horloge fournit plusieurs fonctions pour le travail sur les temps. L'unité de base est la nanoseconde. Plusieurs fonctions sont disponibles pour effectuer des opérations sur les temps :

```
timespec_add(t1, t2)
timespec_sub(t1, t2)
timespec_nz(t)
timespec_lt(t1, t2)
timespec_gt(t1, t2)
timespec_lt(t2, t1)
timespec_ge(t1, t2)
timespec_le(t1, t2)
timespec_eq(t1, t2)
```

2.2. Gestion des tâches

RT-Linux fournit un ordonnanceur par défaut qui possède une interface de programmation conforme à la norme POSIX "pthread". Les fonctions qui ne le sont pas ont un nom avec le suffixe "_np" pour "non-posix". Cet ordonnanceur peut être remplacé par un autre système car il est développé sous forme de module. Il s'appuie sur les couches "bas-niveau" de gestion d'interruptions et d'horloge du module principal de RT-Linux.

```
int pthread_create(pthread_t *p, pthread_attr_t *a, void *(void *), void * x)
```

Cette fonction crée une tâche basée sur la fonction donnée en paramètre en passant le paramètre *x* en premier paramètre de cette fonction. Les attributs de la tâche sont donnés par le paramètre *a*. Une valeur nulle initialise la tâche avec des valeurs par défaut (voir `pthread_attr_init`).

```
int pthread_exit(void * ret)
```

Cette fonction permet à une tâche de se terminer en retournant le code donné en paramètre.

```
int pthread_attr_init(pthread_attr_t *a)
```

Cette fonction permet d'initialiser la structure d'attributs d'une tâche avec les valeurs par défaut. Plusieurs fonctions permettent de positionner les attributs :

```
int pthread_attr_getstacksize(pthread_attr_t *a, size_t * stacksize)
```

```
int pthread_attr_setstacksize(pthread_attr_t *a, size_t stacksize)
```

```
int pthread_attr_getcpu_np(pthread_attr_t *a, int* cpu)
```

```
int pthread_attr_setcpu_np(pthread_attr_t *a, int* cpu)
```

Ces deux dernières fonctions sont des extensions dédiées au support du SMP. Par défaut, les tâches sont créées sur le processeur courant.

```
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param)
```

```
int pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param)
```

Ces fonctions permettent d'initialiser les paramètres d'ordonnancement de la tâche par les attributs de création

```
int pthread_delete_np(pthread_t p)
```

Cette fonction permet de détruire une tâche en cours d'exécution.

```
int pthread_setfp_np(pthread_t p, int flag)
```

Cette fonction permet d'autoriser ou d'interdire l'utilisation des fonctions de calcul en flottant.

```
int pthread_wakeup_np(pthread_t p)
```

Cette fonction permet de réveiller une tâche suspendue.

```
int pthread_suspend_np(pthread_t p)
```

Cette fonction permet de suspendre une tâche.

```
int pthread_wait_np(pthread_t p)
```

Cette fonction permet d'attendre jusqu'à la prochaine interruption de l'horloge.

```
int pthread_make_periodic_np(pthread_t thread, hrtime_t start_time, hrtime_t period)
```

Cette fonction permet de rendre une tâche périodique. Elle démarra à l'heure donnée par `start_time` et fonctionnera avec la périodicité donnée par `period`. Une période nulle génère une seule exécution de la tâche.

2.3. Gestion des MUTEX

Un "mutex" permet de gérer des exclusions mutuelles et permet de protéger des données, des zones d'exécution et de synchroniser des tâches. Il possède deux états, "unlocked" c'est à dire qu'il n'est pas attribué à une tâche donnée, ou "locked" c'est à dire qu'il appartient à une tâche. Une tâche qui tente de verrouiller un mutex pris par une autre tâche est suspendue jusqu'à la libération du mutex par son propriétaire.

```
int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutexattr_t * attr);
```

Cette fonction permet de créer un mutex en spécifiant ses attributs. Si la valeur de `attr` est NULL, le mutex est créé avec des attributs par défaut.

```
int pthread_mutexattr_init(pthread_mutexattr_t *attr)
```

```
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr)
```

Ces fonctions permettent de gérer les attributs d'un mutex. Pour l'instant, il n'y a pas d'options autres que celles par défaut.

```
int pthread_mutex_lock(pthread_mutex_t *mutex)
```

Cette fonction permet de demander le mutex. Si une autre tâche possède le mutex, la tâche appelante est suspendue (si un thread redemande un mutex qu'il possède déjà, il est suspendu).

```
int pthread_mutex_trylock(pthread_mutex_t *mutex)
```

Cette fonction permet de tester l'état du mutex et de le prendre s'il est libre. Cette fonction n'est pas bloquante.

```
int pthread_mutex_unlock(pthread_mutex_t *mutex)
```

Cette fonction libère le mutex et réveille les tâches en attente.

2.4. Gestion des pilotes

Le module "rtl_posixio" permet de mettre en place, coté RT-Linux, une structure de drivers semblable à celle de Linux. Les pilotes seront utilisés grâce aux appels standards (open, read, write, ...).

```
int rtl_register_rtldev(unsigned int mjour, const char * name, struct rtl_file_operations * fops)
```

Cette fonction déclare un pilote en définissant son "major number", le nom du pseudo fichier spécial et la liste des fonctions du pilote. L'ouverture du fichier spécial s'effectue par l'appel open sur le fichier `/dev/nomNN` où "NN" est le "minor number".

```
int rtl_unregister_rtldev(unsigned int mjr, const char * name)
```

Cette fonction permet de décharger un pilote.

2.5. Gestion des FIFOs

La gestion des FIFOs est effectuée par un module de RT-Linux et propose une interface coté processus via les appels systèmes classiques de Linux (open, read, write, close) et coté RT-Linux soit par une interface POSIX soit par des fonctions propres.

```
int rtf_create(unsigned int minor, int size)
```

Cette fonction crée et ouvre le FIFO dont le numéro est donné par "minor" de la taille donnée par "size". Le numéro doit être unique et inférieur au maximum autorisé sur le système ((RTF_NO). Coté Linux, le sFIFOera vu par le fichier spécial /dev/rtfN ou "N" est le numéro.

```
int rtf_destroy(unsigned int minor)
```

Cette fonction détruit le FIFO dont le numéro est donné par "minor".

```
int rtf_get(unsigned int minor, void * buf, int count)
```

```
int rtf_put(unsigned int minor, void * buf, int count)
```

Ces fonctions permettent (respectivement) de lire et d'écrire des données dans le FIFO.

```
int rtf_create_handler(unsigned int minor, int (*handler) (unsigned int fifo))
```

Cette fonction permet d'installer une fonction d'attente sur un FIFO. qui sera exécutée lors d'une écriture ou d'une lecture sur le FIFO.

Chapitre 3. Mise en oeuvre

Ce chapitre présente plusieurs exemples d'utilisation de RT-Linux. Ils sont sensés présenter une vue des différentes possibilités d'utilisation de ce noyau temps réel.

Lors de la phase d'installation de RT-Linux, un fichier `rtl.mk` est généré pour être inclu dans les Makefile. Il contient toutes les définitions des paramètres de compilation des modules.

3.1. Déclenchement périodique

Ce premier exemple est directement tiré de la distribution de RT-Linux. Il s'agit du programme `hello.c` qui affiche un message sur la console système à intervalle régulier.

Le fichier débute par l'inclusion des définitions des paramètres et fonctions des modules utilisés ainsi que par la déclaration de la tâche.

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
pthread_t thread;
void * start_routine(void *arg) {
    struct sched_param p;
    p . sched_priority = 1;
    pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
    pthread_make_periodic_np (pthread_self(), gethrtime(), 500000000);
    while (1) {
        pthread_wait_np ();
        rtl_printf("I'm here; my arg is %x\n", (unsigned) arg);
    }
    return 0;
}
```

La fonction `start_routine` est la fonction principale de la tâche. Elle commence par initialiser le niveau de priorité et passe en mode périodique pour tourner toutes les 500 ms à partir du moment où elle est activée. Elle entre ensuite dans une boucle infinie d'attente du prochain tic d'horloge et d'envoi du message sur la console système.

```
int init_module(void) {
    return pthread_create (&thread, NULL, start_routine, 0);
}
```

La fonction `init_module` est appelée lors du chargement du module par la commande **insmod**. Elle se contente de créer la tâche avec les paramètres par défaut.

```
void cleanup_module(void) {
    pthread_delete_np (thread);
}
```

La fonction `cleanup_module` est appelée lors du déchargement du module par la commande **rmmod**. Elle détruit la tâche en cours.

3.2. Producteur-consommateur

Le but de cet exemple est de construire une application avec un dialogue entre un module RT-Linux et un processus Linux. Le module RT-Linux est le producteur de données que le processus standard consomme. Le producteur tourne périodiquement et envoie sur le FIFO l'intervalle de temps entre deux appels.

Le fichier débute par l'inclusion des définitions des paramètres et fonctions des modules utilisés ainsi que par la déclaration de la tâche. On définit le numéro du FIFO qui sera utilisé par ce module.

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#include <rtl_fifo.h>
#define FIFO_NUM          1
#define FIFO_SIZE        2048

pthread_t thread;
void * Generator(void *arg)
{
    int ret;
    struct sched_param p;
    hrtime_t      time_prec;
    hrtime_t      time;
    long          delta;
    p.sched_priority = 1;
    ret = pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
    if (ret < 0)
    {
        printk("test1 Generator() pthread_setschedparam returns %d\n", ret);
        return ret;
    }
    ret = pthread_make_periodic_np (pthread_self(), gethrtime(), 500000000);
    if (ret < 0)
    {
        printk("test1 Generator() pthread_make_periodic_np returns %d\n", ret);
        return ret;
    }
    time_prec = gethrtime();
    while (1) {
        pthread_wait_np ();
        time = gethrtime();
        delta = time - time_prec;
        if (rtf_put(FIFO_NUM, (char *) &delta, sizeof(long)) < 0)
        {
            printk("test1 Generator() rtf_put returns %d\n", ret);
        }
    }
}
```

```
        return ret;
    }
    time_prec = time;
}
return 0;
}
```

La fonction `Generator` ressemble à celle de l'exemple précédent. La différence principale tient dans l'envoi d'informations dans le FIFO par la fonction `rtf_put`.

```
int init_module(void)
{
    int ret;
    ret = rtf_create(FIFO_NUM, FIFO_SIZE);
    if (ret < 0)
    {
        printk("test1 init_module() rtf_create returns %d\n", ret);
        return ret;
    }
    ret = pthread_create (&thread, NULL, Generator, 0);
    if (ret < 0)
    {
        printk("test1 init_module() pthread_create returns %d\n", ret);
        return ret;
    }
    return 0;
}
```

La fonction d'initialisation du module crée le FIFO et la tâche.

```
void cleanup_module(void) {
    rtf_destroy(FIFO_NUM);
    pthread_delete_np (thread);
}
```

La fonction de destruction du module détruit le FIFO et la tâche.

3.3. Pilote de périphérique

Cet exemple montre la mise en oeuvre d'un pilote de périphérique, coté RT-Linux, utilisant l'interface POSIX pour les appels de fonctions. Pour ne pas surcharger la présentation, seul un squelette des fonctions est présenté. Pour un driver totalement opérationnel, les sources de "rt_com" sont disponibles dans la distribution de RT-Linux.

```
int init_module( void )
{
    // Initialisation du matériel, test de présence, etc..
    . . .
}
```

```
    // Initialisation de la gestion des interruptions et validation de la recep-
tion d'IT
    rtl_request_irq( NU_IRQ, driver_isr);
    rtl_hard_enable_irq( NU_IRQ);
    // Enregistrement du driver dans le module POSIX
    if (rtl_register_chrdev (DRIVER_MAJOR, "driver", &driver_fops)) {
        printk ("Driver : unable to get RTL major %d\n", DRIVER_MAJOR);
        return -EIO;
    }
    return(0);
}
```

La fonction `init_module` initialise les variables internes du driver, installe la fonction de gestion des interruptions et enregistre le driver dans la structure POSIX. Les valeurs du numéro d'interruption et du major number doivent être fixées en définition. La structure `driver_fops` doit être initialisée en fonction des appels implémentés.

```
void cleanup_module( void )
{
    rtl_unregister_chrdev(DRIVER_MAJOR, "driver");
    rtl_free_global_irq(NU_IRQ);
    // Liberation de l'ensemble des ressources allouées
    .
    return 0;
}
```

La fonction `cleanup_module` libère les ressources allouées par `init_module`.

La structure de définition du driver rassemble les pointeurs sur les fonctions implémentant les appels systèmes :

```
static struct rtl_file_operations driver_fops = {
    driver_llseek,
    driver_read,
    driver_write,
    NULL,
    driver_ioctl,
    driver_mmap,
    driver_open,
    driver_release
};
static int driver_open (struct rtl_file *filp)
{
    // Operations sur l'appareil donné par filp->f_minor.
    return 0;
}
```

La fonction `open` reçoit la structure `rtl_file` initialisée par RT-Linux et peut réaliser des initialisations spécifiques en fonction du minor number.

```
static int rtl_rt_com_release (struct rtl_file *filp)
{
```

```
    // Operations sur l'appareil donné par filp->f_minor.  
    return 0;  
}
```

La fonction `release`, appelée à la fermeture du pilote, reçoit la structure `rtl_file` initialisée par RT-Linux et peut réaliser des opérations spécifiques en fonction du `minor number`.

```
static ssize_t driver_write(struct rtl_file *filp, const char *buf, size_t count, loff_t* ppos)  
{  
    int cnt = 0;  
    // Vérifications des paramètres et du contexte  
    .  
    // Masquage des interruptions  
    rtl_no_interrupts( state );  
    // Ecriture sur le hardware  
    .  
    // Validation des interruptions  
    rtl_restore_interrupts( state );  
    return cnt;  
}
```

La fonction d'écriture du driver peut masquer les interruptions avant d'effectuer des opérations sur le matériel. Elle retourne le nombre d'octets écrits.

```
static ssize_t driver_read(struct rtl_file *filp, char *buf, size_t count, loff_t* ppos)  
{  
    int cnt = 0;  
    // Vérifications des paramètres et du contexte  
    .  
    // Masquage des interruptions  
    rtl_no_interrupts( state );  
    // Ecriture sur le hardware  
    .  
    // Validation des interruptions  
    rtl_restore_interrupts( state );  
    return(cnt);  
}
```

La fonction de lecture du driver peut bloquer les interruptions pour accéder au matériel. Elle doit retourner le nombre d'octets lus.

```
loff_t driver_llseek(struct rtl_file *filp, loff_t offset, int whence)  
{  
    return pos;  
}
```

La fonction `driver_llseek` permet d'effectuer une opération de positionnement sur le périphérique, si cette opération à un sens pour celui-ci.

```
int driver_mmap(struct rtl_file *filp, void * start, size_t length, int prot, int flags, off_t off-
set, caddr_t * res)
{
    return pos;
}
```

La fonction `driver_mmap` permet de présenter le périphérique comme une zone mémoire au processus utilisateur.

```
unsigned int driver_isr( unsigned int nu_irq, struct pt_regs *r )
{
    // Traitement de l'interruption
    .
    // Revalidation des interruptions
    rtl_hard_enable_irq(nu_irq);
    return 0;
}
```

La routine de gestion des interruptions reçoit en paramètre le numéro d'interruption et la structure des registres. Le code de gestion doit être aussi concis que possible pour limiter le traitement en mode masquage d'interruption.

Appendice A. Bibliographie

Les pages de manuel et les sources plus ces quelques documents :

- Real-Time Linux (RT- Linux) by Ismael Ripoll : <http://www.nl.linuxfocus.org/English/May1998/article4.html>
- RTLinux Version Two Viktor Yodaiken and Michael Barabanov VJY Associates LLC

