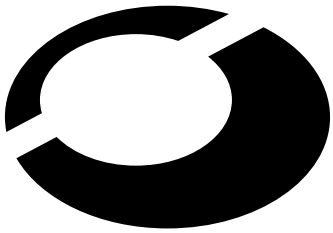
www.alcove.com

RTLinux

Stéphane List, Nicolas Ferre
slist@tuxfamily.org, nferre@free.fr

version 1.22



l'informatique est libre

Alcôve

Alcôve
Centre Paris Pleyel
153 bd Anatole France
93200 Saint-Denis, France
Tél. : +33 1 49 22 68 00
Fax : +33 1 49 22 68 01
E-mail : alcove@fr.alcove.com, Toile : www.alcove.com

Table des matières

Définition et concepts de RTLinux

Le temps réel...

FAQ

`http ://www.faqs.org/faqs/realtime-computing/faq/`

Définition du IEEE :

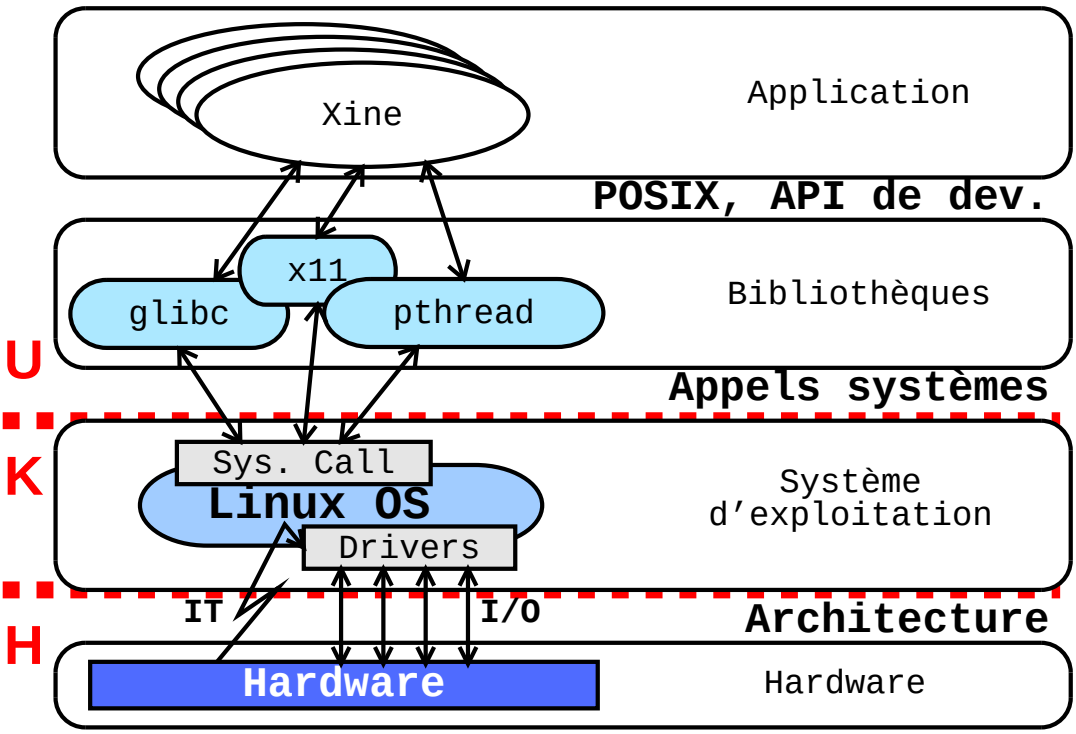
"Un système temps réel est un système dont le temps de réponse est aussi important que la qualité de fonctionnement."

Définition POSIX :

"Le temps réel est la capacité à répondre à une sollicitation en un temps déterminé pour produire une réaction appropriée."

Multitude de définitions du temps réel qui peut être qualifié de mou/dur, critique...

Système GNU/Linux : interfaces et couches d'abstractions

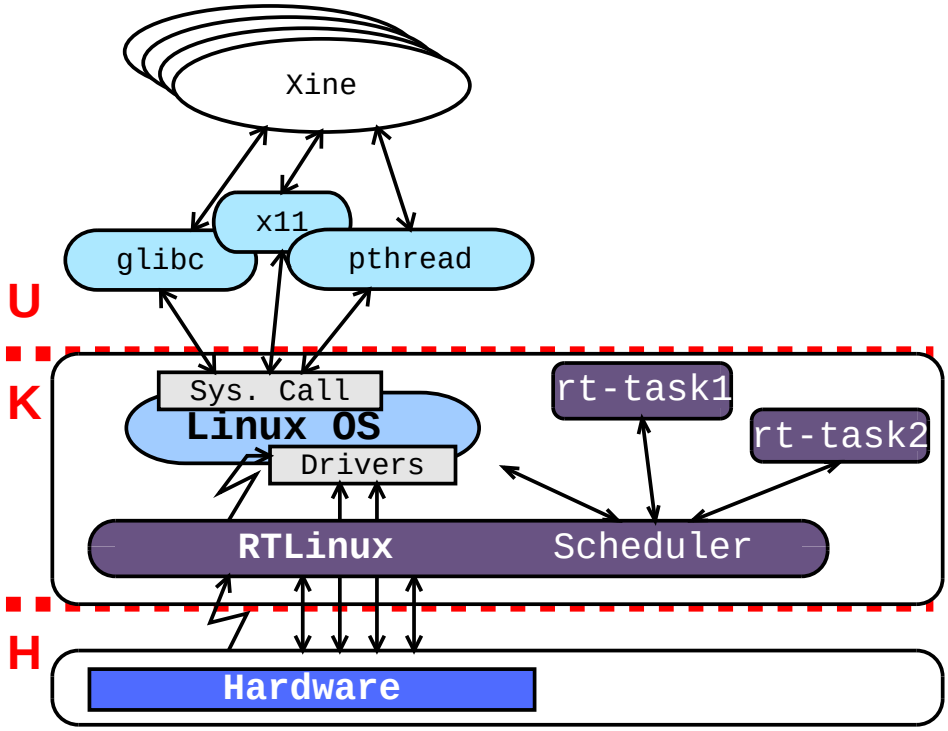


RTLinux : un micro-noyau

RTLinux se présente sous la forme d'un micro-noyau prenant place entre la machine réelle et le noyau Linux.

- Multitâche **et** totalement préemptif ;
- SMP sur x86, tourne aussi sur PPC, Alpha (version OpenRTLinux) ;
- + AMD Elan, MIPS, Synergy MicroSystems, IBM Walnut, StrongARM, RTLinux/BSD (version propriétaire) ;
- Ordonnanceur (scheduler) ;
- IPC et système de synchronisation entre les tâches ;
- Accès direct au matériel ;
- Espace d'adressage du noyau Linux / pas de protection mémoire ;
- Module de "debug" et de trace du système.

RTLinux : Architecture conceptuelle



Définition et concepts de RTLinux

Alcôve - RTLinux

Principes : Les tâches

- Linux est la tâche temps réel de plus faible priorité ;
- Création de tâche depuis un contexte Linux (init_module) ou RTLinux ;
- Attributs d'ordonnancement, taille de pile, utilisation de FPU, etc ;
- Dialogue entre tâches temps réel et processus Linux par FIFOs, mémoire partagée ou interruptions virtuelles (signaux ou soft IT) ;

Principes : les interruptions (1/2)

– Code de masquage/démasquage des interruptions réécrit ;

Le cli Linux "désactive la montée d'IT virtuelles"
Le sti Linux "réactive la montée d'IT virtuelles, émule les interruptions en attente"

– Interruptions matérielles (hard IT) interceptées puis éventuellement routées vers Linux par la couche temps réel ;
– Générations d'interruptions virtuelles (soft IT) destinées à Linux ;
– Linux ne peut intervenir sur les interruptions matérielles.


Définition et concepts de RTLinux
Alcôve - RTLinux

Principes : les interruptions (2/2)

– Sur montée d'interruption :
  - *Si un handler RTLinux a été initialisé : l'appeler*
  - *Si un handler Linux a été initialisé ET RTLinux inactif ET IT virtuelles activées : appeler le handler Linux*
  - *Sinon : marquer l'interruption comme étant en attente*

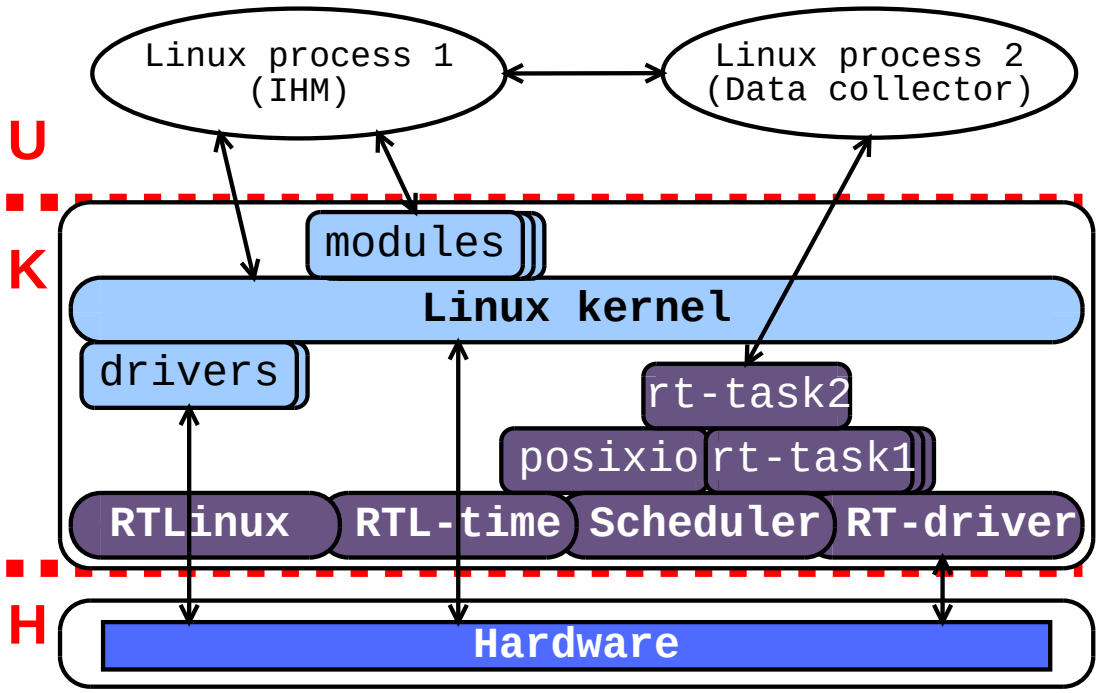
Outils et interfaces

RTLinux est structuré comme partie centrale minimale sur laquelle vient se charger une collection de modules qui fournissent des services optionnels ou des niveaux d'abstraction.

- `rtl, rtl_time, rtl_sched` : RTLinux, timer, scheduler ;
- `mbuff` : Shared Memory ;
- `rtl_fifo` : FIFOs ;
- `rtl_posixio` : API POSIX pour le système de fichiers ;
- `rtl_debug`, `rtl_tracer` : débogage ;
- ...

=>Système modulaire et extensible.

Architecture modulaire



Besoin d’un nouveau modèle de programmation

Découpage de l’application :

"Si un service est intrinsèquement non temps réel, il doit être fourni par Linux et non par les modules RTLinux."

Laisser faire à Linux ce qu’il sait bien faire : GUI, base de données, drivers non temps réel...

Gestion de la mémoire : allocation statique / pas de protections...

Communications non-bloquantes côté temps réel.

Utilisation de modules chargeables / API spécifique.


Définition et concepts de RTLinux
Alcôve - RTLinux

Installation de RTLinux (1/2)

- Télécharger RTLinux ;
- Chercher les versions de Linux supportées par RTLinux ;
- Télécharger Linux ;
- Patcher Linux avec RTLinux ;
- Configurer / Compiler / Installer le noyau ;
- Rebooter.

Installation de RTLinux (2/2)

- Configurer / Compiler / Installer RTLinux ;
- Faire tourner les exemples (`make test`) ;
- `rtl-config` permet de vérifier la config ;
- `rtlinux` est un script pour charger les modules RTLinux ;
- `insrtl` et `rmrtl` charge et décharge les modules RTLinux.

Approche micro-noyau : critères de choix

- Avantages :
  - Comportement temps réel critique ;
  - Modularité, scalabilité ;
  - Compacité du code et du patch noyau (< 100 ko/< 1000 lignes) ;
    - Version récente du noyau Linux (dernières innovations) ;
    - Peu de bugs, correction rapide (mailing lists), testé par de nombreux spécialistes.
- Inconvénients :
  - Modification des drivers pour un comportement temps réel ;
  - Apprentissage d'une API (facilité par la compatibilité POSIX) ;
  - Difficultés de débogage (espace noyau).

Ordonnancement

Ordonnanceurs

RTLinux fonctionne en mode Uni-Processeur et SMP (Symetrical Multi-Processor)

En mode SMP une tâche est affectée à un processeur donné. `pthread_attr_setcpu_np()` permet d'assigner une tâche à un CPU particulier. Permet aussi de réserver un CPU à RTLinux

Remarque : Attention lors de la compilation du noyau Linux de ne sélectionner SMP que si la machine est bien SMP, utiliser le bon type de CPU, désactiver l'APM.

Politiques d’ordonnancement

Les créateurs de RTLinux voulant garder un système simple n’ont implémenté qu’un seul type d’ordonnancement :
– tâches à *priorités fixes* ;
– choix exclusivement par ordre de priorité ;
– argument de politique `SCHED_FIFO` pour compatibilité.

Mais des initiatives de la communauté RTLinux visent à ajouter d’autres politiques d’ordonnancement :
– `SCHED_EDF_NP` soit Earliest Deadline First ;
– choix des tâches d’échéance plus proche sur un niveau de priorité ;
– utilisation d’algorithme Stack Resource Protocol (SRP) pour la gestion des sémaphores (évite l’inversion de priorité).

Priorité des tâches

Les priorités min et max que peuvent avoir les tâches RTLinux dépendent de la politique d’ordonnancement, pour connaître ces valeurs, il faut utiliser les fonctions :
– `int sched_get_priority_max()` ;
– `int sched_get_priority_min()`.

Tâches (threads)

Déclaration d'une tâche

```
#include <pthread.h>
int  pthread_create (pthread_t *thread, // Tâche
pthread_attr_t *attr, //Attributs de la tâche
void* (*start_routine)(void*), // Routine de la
tâche
void* arg); //paramètre d'appel de la routine
```

Modification des attributs d’une tâche

En contexte Linux :
– `pthread_attr_init(), pthread_attr_destroy();`
– `pthread_attr_[get|set]schedparam();`
– `pthread_attr_[set|get]cpu_np();`
– `pthread_attr_[set|get]stacksize();`
– `pthread_attr_setfp_np()`.

Ou en contexte RTLinux :
– `pthread_[get|set]schedparam();`
– `pthread_setfp_np()`.

Nombres flottants dans une tâche RTLinux

L’émulation des nombres flottants présent dans le noyau Linux ne peut être utilisés dans une tâche RTLinux

RTLinux utilise la FPU présente sur x86 (> 486 DX) et PPC

Sauvegarde de registres supplémentaires => changement de contexte plus lent.

Possibilité d’utiliser une librairie mathématique (ajouter `-lm` à l’édition de liens)

Suppression d'une tâche

- `pthread_cancel()` envoi de notification de fin d'exécution ;
- puis `pthread_join()` attente jusqu'à la fin du thread désigné ;
- `pthread_delete_np()` rassemble ces deux fonctions ;
- `pthread_setcancelstate()` :
  - `PTHREAD_CANCEL_ENABLE` (par défaut) ;
  - `PTHREAD_CANCEL_DISABLE` ;
- `pthread_setcanceltype()` : acceptation de terminaison
  - immédiate (`PTHREAD_CANCEL_ASYNCHRONOUS`) ;
  - à certains point de l'exécution (`PTHREAD_CANCEL_DEFERRED`) (par défaut) ;
- `pthread_testcancel()` : point de possible fin d'exécution du thread ;

Programmation des tâches périodiques

- `pthread_make_periodic_np()` rend une tâche périodique ou modifie la période "à la volée" ;
- `pthread_wait_np()` rend la main jusqu'à la prochaine période.

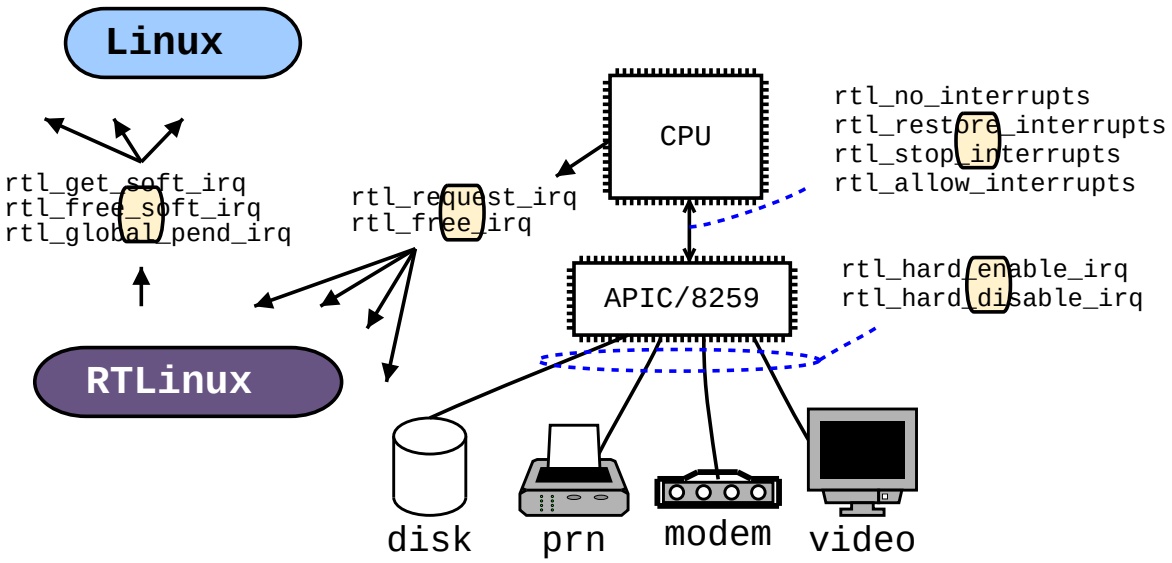
Programmation des tâches a-périodiques ou sporadiques

- `pthread_suspend_np()` endort une tâche ;
- `pthread_wakeup_np()` réveille une tâche ;
- envoie respectivement les signaux `RTL_SIGNAL_SUSPEND` et `RTL_SIGNAL_WAKEUP`.

Permet de construire une tâche exécutée sur interruptions : le handler d'interruptions appelle `pthread_wakeup_np()`.

Interruptions

Gestion des interruptions



Masquage au niveau du [A]PIC :

- `rtl_hard_disable_irq()` ;
- `rtl_hard_enable_irq()`.

Masquage au niveau du CPU (propre à un CPU pour le SMP) :

- `rtl_no_interrups()` sauvegarde des registres d'état du CPU ;
- `rtl_restore_interrupts()` ;
- `rtl_stop_interrupts()` simples cli/sti ;
- `rtl_allow_interrupts()`.

Installer un handler d'interruptions

Interruptions matérielles (hard IT) :
- `rtl_request_irq()` : installe un handler pour une IRQ ;
- `rt_free_irq()` : dés-installe un handler d'IRQ.

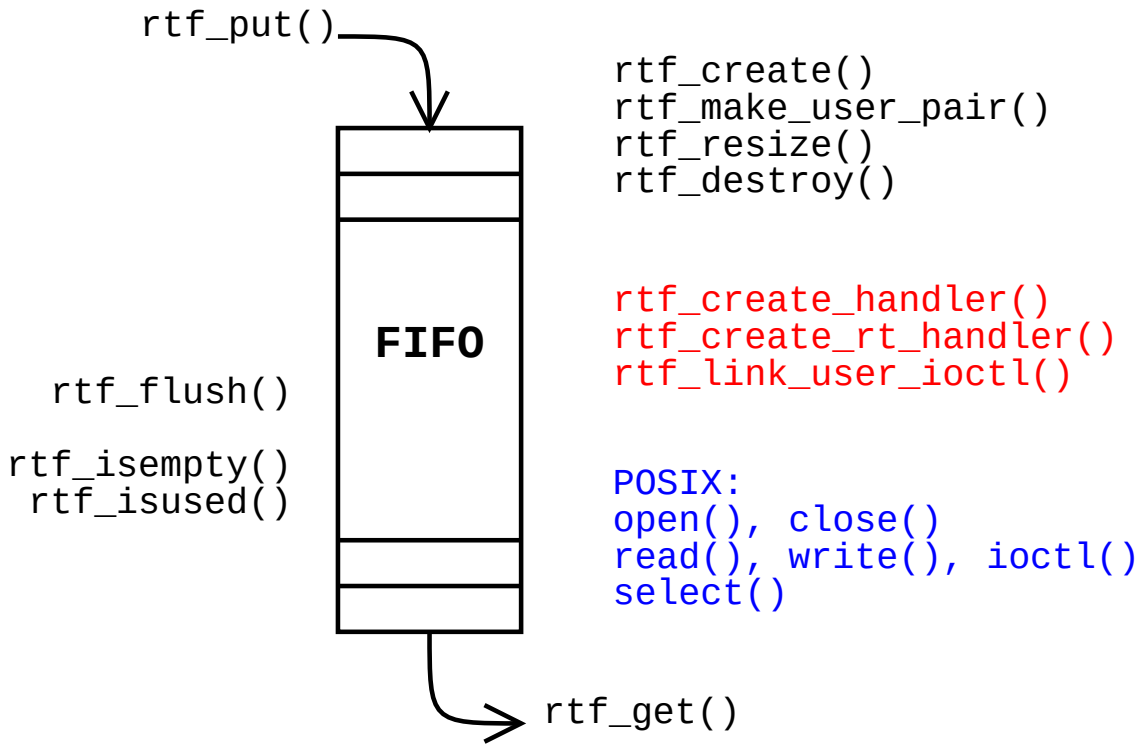
Interruptions virtuelles (communication avec le noyau Linux) :
- `rtl_get_soft_irq()` : installe un handler sur une IRQ virtuelle ;
- `rt_free_soft_irq()` : dés-installe le handler.

Déclencher la remontée d'interruptions vers le noyau Linux :
- `rtl_global_pend_irq()` partage d'IRQ (Linux/RTLinux).

IPC : outils de communication inter processus

Communication entre RTLinux et Linux

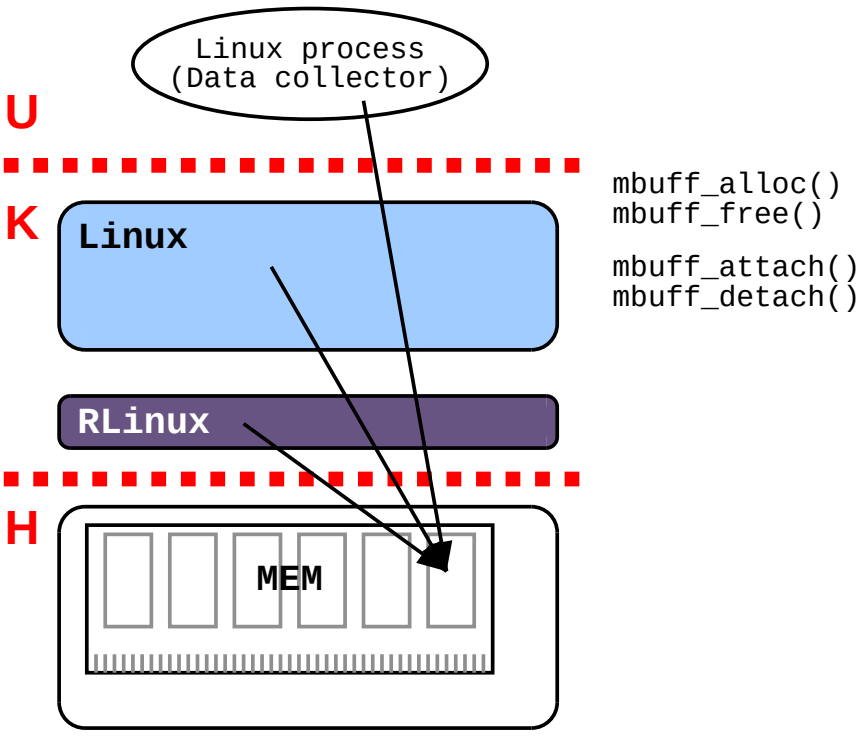
FIFO : First In First Out



FIFO : caractéristiques

- créées/détruites en contexte Linux ;
- unidirectionnelles ou utilisation de paires pour échanges bidirectionnel (`rtf_make_user_pair()`) ;
- accessibles dans le système de fichiers Linux : /dev/rtfX comme périphérique en mode caractère (major 150 minor 0-63) ;
- les FIFOS doivent être ouvertes en mode `O_NONBLOCK` par les tâches RTLinux (si I/O POSIX) ;
- Appel de routines sur évènements Linux/RTLinux (`rtf_create_handler()`/`rtf_create_rt_handler()`).

Mémoire partagée



Mbuff : caractéristiques

- mémoire virtuelle (utilise `vmalloc()`) ;
- créées/détruites en contexte Linux ;
- zones mémoires nommées ;
- compteur d'utilisation ;
- Informations dans `/proc/mbuff` ;
- Portable : Linux, RTLinux, RTAI.

Mémoire partagée : allocation au démarrage

Ancienne méthode permettant de partager la mémoire entre les processus utilisateurs Linux et les tâches RTLinux.

- mémoire physique ;
- allocation permanente au boot du système `append="mem=31m"` dans `lilo.conf` ;
- accès depuis Linux : projection dans l'espace d'adressage du processus de l'adresse physique (`/dev/mem`) ;
- accès depuis RTLinux : par pointeur sur la zone mémoire physique.

Exemple :

```
#define BASE_ADDRESS (31 * 0x100000)
MY_STRUCT *ptr;
```

Dans Linux

```
fd = open ("/dev/mem", O_RDWR);
ptr = (MY_STRUCT *) mmap (0, sizeof(MY_STRUCT),
PROT_READ | PROT_WRITE,
MAP_FILE | MAP_SHARED,
fd, BASE_ADDRESS);
```

Dans RTLinux

```
ptr = (MY_STRUCT *) BASE_ADDRESS;
```


Mécanismes de synchronisation

Exclusion mutuelle(1/3)

– synchronisation ;
– intégrité des données partagées.

Leur utilisation peut être évitée : réduction au minimum des sections critiques, exécutées interruptions masquées.
– `pthread_mutex_init()`, `pthread_mutex_destroy()` ;
– `pthread_mutex_lock()` "P" ;
– `pthread_mutex_trylock()` si le verrou est pris : pas de bloquage, retour de code d'erreur ;
– `pthread_mutex_timedlock()` sémaphore avec timer ;
– `pthread_mutex_unlock()` "V" ;
– De nombreuses fonctions pour fixer les attributs du verrou.

Exclusion mutuelle (2/3)

Types de verrous :
– `PTHREAD_MUTEX_NORMAL = PTHREAD_MUTEX_DEFAULT` ;
– `PTHREAD_MUTEX_SPINLOCK_NP` (verrou à attente active -SMP-).

Portée des sémaphores d'exclusion mutuelle (mutex) :

La norme POSIX définie deux sortes de portée pour les verrous `PTHREAD_PROCESS_SHARED` et `PTHREAD_PROCESS_PRIVATE`.

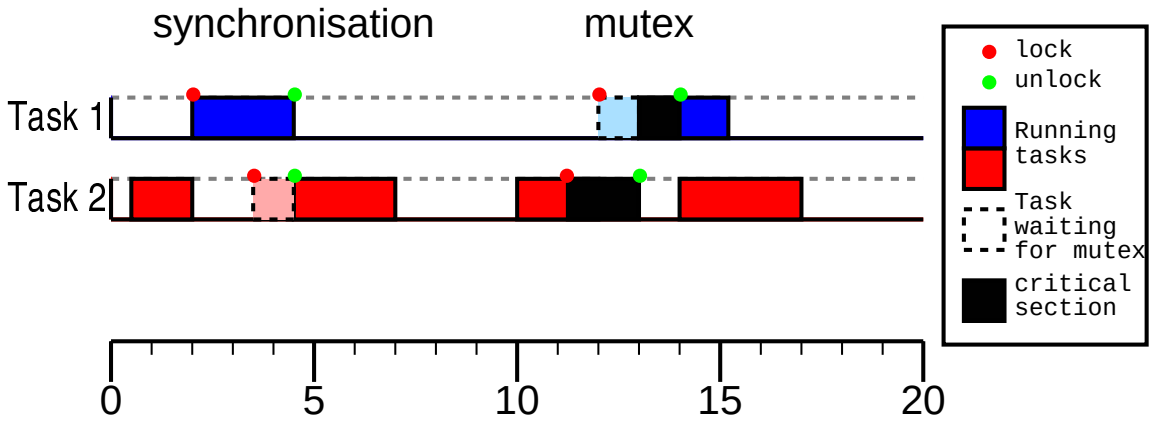
RTLinux n'implémente pas la gestion de ces attributs.

Exclusion mutuelle (3/3)

Protocole d'action des mutex :

– `PTHREAD_PRIO_INHERIT` Priority Inheritance Protocol (PIP) (non implémenté). La tâche détenant le verrou hérite de la priorité de la tâche de plus forte priorité en attente sur le mutex ;
– `PTHREAD_PRIO_NONE` Priorités statiques ;
– `PTHREAD_PRIO_PROTECT` Ceiling Semaphore Protocol (CSP). La tâche détenant le verrou hérite de la priorité attribuée à la ressource contrôlée par celui-ci ;
– `PTHREAD_PRIO_SRP` Stack Resource Policy (SRP). La tâche qui commence son exécution ne sera pas bloquée jusqu'à sa fin. Utilisé avec la politique d'ordonnancement EDF.

Gestion des verrous



API complémentaires : Variables condition et sémaphores

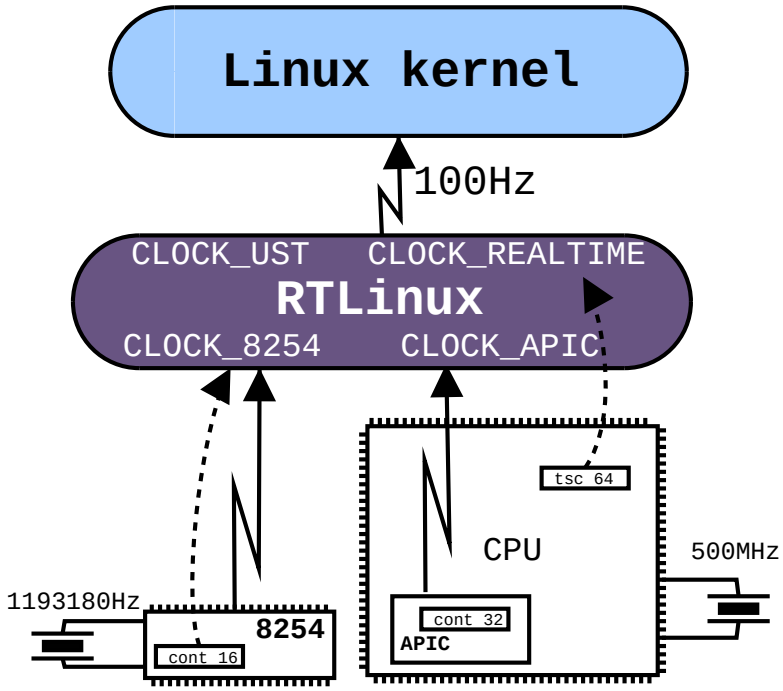
Permet d'étendre les possibilités de synchronisation : sémaphores non binaires ou "à compteur"

- `pthread_cond_init()`, `pthread_cond_destroy()` ;
- `pthread_cond_wait()`, `pthread_cond_timedwait()` ;
- `pthread_cond_signal()` ;
- `pthread_cond_broadcast()` réveille toutes le tâches en attente sur une variable condition.

- `sem_init()`, `sem_destroy()` ;
- `sem_getvalue()` ;
- `sem_wait()`, `sem_timedwait()`, `sem_trywait()` ;
- `sem_post()`.

Le temps

Gestion des horloges (clocks)



Gestion des horloges (1/2)

RTLinux dispose de 3 horloges logiques (POSIX) :

- `CLOCK_MONOTONIC` temps écoulé depuis le boot du système ;
- `CLOCK_REALTIME (= CLOCK_RTL_SCHED)` horloge utilisée par l'ordonnanceur (et fonctions `pthread_make_periodic_np()`, `nanosleep()`, `sem_timedwait()`, etc.) ;
- `CLOCK_GPOS` converge vers l'horloge du système : sujette aux ajustements NTP.

Gestion des horloges (2/2)

Les horloges physiques disponibles sont le reflet de l'architecture de la machine (ici x86) :

– `CLOCK_8254` opérations d'E/S sur la puce très lente (bus ISA)
  Fréquence d'horloge : 1 193 180 Hz ;
– `CLOCK_APIC` Sur systèmes SMP et UP récents : utilise le le tsc (time stamp counter) des processeurs : le référentiel de temps est donc propre à chacun d'eux
  Fréquence d'horloge du processeur.

Fonctions d'accès aux horloges

Quelle heure est il ?

```
int clock_gettime (clockid_t clock_id, struct
timespec *ts);
```

```
hrtime_t clock_gethrtime (clockid_t clock);
```

```
struct timespec {
time_t tv_sec; /* seconds */
long tv_nsec; /* nanoseconds */
};
```

Routines de conversion :

```
timespec_to_ns(), timespec_from_ns(), hrt2ts()
```

Gestion de minuteries (timers)

2 modes de programmation :

– `RTL_CLOCK_MODE_ONESHOT` :
  - + on atteint la précision du matériel ;
  - - reprogrammation à chaque interruption.
– `RTL_CLOCK_MODE_PERIODIC` :
  - + programmation une fois pour toute ;
  - - précision en fonction de la résolution choisie ;
  - - surcharge du système par interruptions ;
  - +/- moins flexible mais plus efficace.

Mode fixé par la fonction : **rtl_setclockmode()** .

API d’utilisation de minuteries

– attente active :
  - `rtl_delay()` (en nanosecondes).
– attente sur timer (non active) - ne pas utiliser depuis une routine de traitement d’interruptions :
  - `usleep()` (en microsecondes) ;
  - `clock_nanosleep()` (en nanosecondes) peut être interrompue par un signal ;
  - `nanosleep()` (en nanosecondes) utilise `CLOCK_REALTIME`.

Interruption d'horloge

Interruption intimement liée au fonctionnement des systèmes RTLinux et Linux (HZ)

Manipulation prohibée à l'aide des routine de traitement des IT (`rtl_request_irq()`) => mise en place d'IT virtuelle
- `rtl_setclockhandler()`;
- `rtl_unsetclockhandler()`.

Utilisation impossible avec le scheduler qui installe son propre handler d'IT horloge

Entrées / Sorties

Accès à la mémoire physique et aux ports

Remarques

Ceci n’est qu’une brève introduction rappelant ces concepts propres au noyau Linux.

Pour une approche plus détaillée, se référer au cours Alcôve

"Noyau Linux et pilotes de périphériques".

Fonctions d’accès direct aux ports

Plages d’adresses utilisées par les périphériques présents sur le bus d’E/S (ex : registres de contrôle) ;

On accède à ces ports depuis le noyau grâce aux fonctions :
– `in{b,w,l}()`/`out{b,w,l}()` : lit/écrit 1, 2 ou 4 octets consécutifs sur un port d’E/S ;
– `in{b,w,l}_p()`/`out{b,w,l}_p()` : lit/écrit 1, 2 ou 4 octets consécutifs sur un port d’E/S et fait une pause (une instruction) ;
– `ins{b,w,l}()`/`outs{b,w,l}()` : lit/écrit des séquences de 1, 2 ou 4 octets consécutifs sur un port d’E/S.

Projection dans l’espace d’adressage du noyau

En contexte Linux, on utilise `ioremap()` pour mapper une plage d’adresses physiques sur la plage d’adresses linéaires du noyau.

On accède ensuite à la mémoire partagée des E/S grâce aux fonctions suivantes :
– `read{b,w,l}()`/`write{b,w,l}()` : lit/écrit respectivement 1, 2 ou 4 octets consécutifs dans de la mémoire d’E/S ;
– `memcpy_{from,to}io()` : lit/écrit un bloc d’octets consécutifs dans de la mémoire d’E/S ;
– `memset_io()` : remplit une zone de mémoire d’E/S avec une valeur fixe ;
– `virt_to_bus()`/`bus_to_virt()` : traduction entre adresses virtuelles linéaires et adresses réelles sur le bus.

Drivers

Portage de drivers Linux vers RTLinux

Un driver Linux est préempté par toute action en espace RTLinux. Pour le rendre déterministe, il faut l’adapter à l’environnement RTLinux :

– gestion des interruptions ;
– protection des ressources / synchronisation ;
– échanges de données entre espace utilisateur et noyau ;
– interface "fichiers" de style POSIX.

Module posixio

Utilisation de librairies de compatibilités

– DPI (Driver Programming Interface) ;
– Comedi (Control and Measurement Device Interface).

Comedi : Interface de développement de drivers *RT[Linux|AI]* pour cartes d'acquisition de données

– nombreuses cartes déjà supportées (convertisseurs A/D D/A, entrées/sorties numériques, capteurs température, etc.) ;
– interface modulaire ;
– applicatif indépendant vis à vis du matériel.

Driver de liaison série : rt_com

La liaison série se programme avec l'API POSIX (open, write, read, ioctl) ou les fonctions suivante :

– rt_com_write ;
– rt_com_read ;
– rt_com_setup.

Facilités d'utilisation

Débogguer RTLinux

Utilisation de traces

- Option CONFIG_RTL_TRACER à sélectionner lors de la configuration de RTLinux ;
- Module `rtl_tracer.o` ;
- Traces déjà existantes dans le coeur de RTLinux ;
- Définition du champs d'application : `rtl_trace_settracemask` ;
- Traçage manuel : `rtl_trace()`, `rtl_trace2()` ;
- Définition de nouvelles catégories : `rtl_trace_seteventclass` et `rtl_trace_seteventname` ;
- Traces dumpées dans une mémoire partagées, à récuperer avec `tracer.c` et `symresolve`.

Le déboggeur intégré à RTLinux

- Option CONFIG_RTL_DEBUG et CONFIG_RTL_DEBUGGER à sélectionner lors de la configuration de RTLinux ;
- Module `rtl_debug.o` ;
- Mettre un breakpoint initial : `breakpoint()` ;
- Utiliser `gdb` + macros additionnelles (ou `ddd`...) ;
- Possibilité de *step* , *next* , *breakpoint* etc...

Projets connexes

miniRTL

MiniRTL est une implémentation réduite de RTLinux qui tient sur une disquette de 1.44Mo. miniRTL a les spécificités suivantes :

– Linux 2.2.13, RTLinux 2.0 ;
– support réseau (ethernet, slip, plip) ;
– ssh/scp, sunrpc ;
– mail sortant ;
– mini serveur web avec support des CGI.

RTIC-Lab

RTIC-Lab est un projet en GPL qui permet d’avoir une interface graphique (gtk) pour le contrôle d’une application temps réel RTLinux. RTIC-Lab permet de s’interfacer avec des cartes d’entrées analogiques/digitales.

Licence

Rappels

GPL : GNU General Public License.

LGPL : GNU Lesser General Public License.

GPL et LGPL : droit et obligation de redistribution des sources modifiées.

GPL : "contamination" lors du "link" : pas de propriétaire (ou autre licence).

LGPL : pas de "contamination" : propriétaire (ou autre license) possible.

RTLinux

Brevet US (U.S. Patent No. 5,995,745).

Deux versions :
– OpenRTLinux : licence GPL ;
– RTLinux/Pro : licence propriétaire.

Le brevet est licencié gratuitement pour toute utilisation de :
– OpenRTLinux non modifié ET
– "code" entièrement GPL.

Code propriétaire applicatif avec OpenRTLinux possible mais pas clair...

RTAI

Licence GPL pour le coeur de RTAI.

Licence LGPL pour utilitaires/librairies.

Brevet RTLinux applicable ?

Travaux Pratiques

Exercice 1

Installer RTLinux.

Écrire un module RTLinux qui écrit *Hello World* dans le fichier log, depuis un contexte noyau Linux et depuis une tâche RTLinux.

Exercice 2

Écrire un module RTLinux faisant tourner une tâche PERIODIQUE (T = 1s) qui écrit *Hello World* 10 fois dans le fichier log.

– Fixer les paramètres du scheduler en contexte RTLinux ;
– Fixer les paramètres du scheduler à l'initialisation de la tâche temps réel (en utilisant `pthread_attr_*()`).

Exercice 3

Écrire un module RTLinux faisant tourner une tâche PERIODIQUE (T = 1s) qui calcule et affiche le temps écoulé entre deux activations de la tâche.

Même exercice en utilisant un mode de programmation au coup par coup (one-shot).

Comparer les performances des horloges périodiques et coup par coup à l'aide de l'exemple `measurements`.

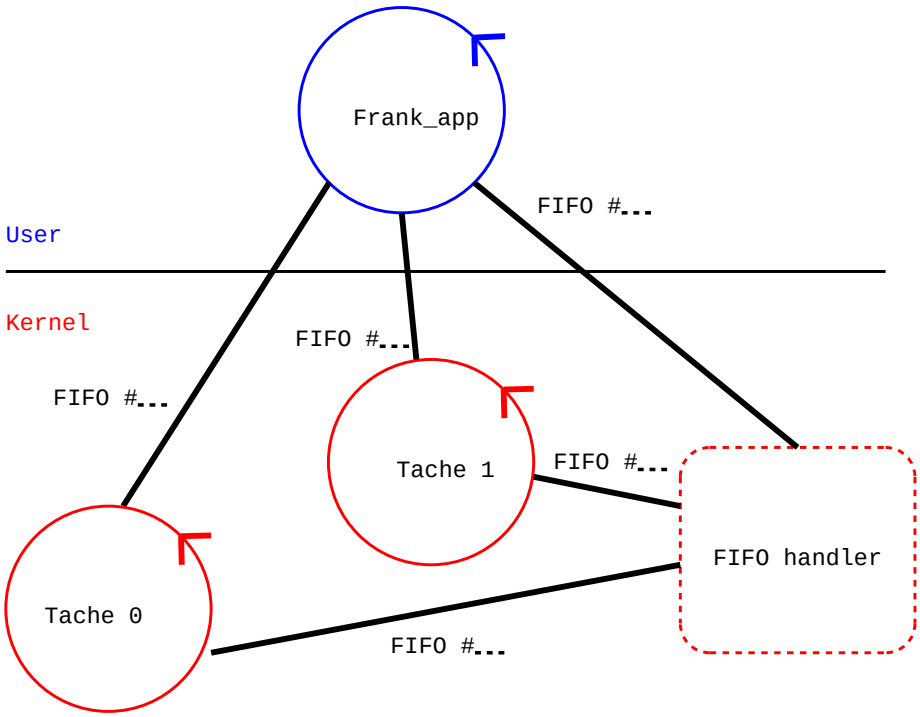
Exercice 4

Écrire un module RTLinux avec une tâche périodique qui écrit un compteur de 1 à 10 dans une FIFO.

Et écrire un programme en mode utilisateur qui va lire la FIFO.

Que se passe-t-il si on appelle 2 fois la fonction de création de FIFO pour la même FIFO (cf. code source) ?

Exercice 5



TP : Retrouvez les numéros des FIFOS
et le sens de passage des données

Exercice 6

Écrire un module RTLinux qui écrit une valeur dans une mémoire partagée.

Et écrire un programme en mode utilisateur qui va lire la mémoire partagée.

Exercice 7

Écrire un module RTLinux qui compte les interruptions produites par le clavier et qui écrit la valeur du compteur dans une mémoire partagée.

Et écrire un programme en mode utilisateur qui va lire la mémoire partagée.

(Écrire le module en utilisant *rt_request_linux_irq* puis avec *rt_request_global_irq*)

Remerciements / Références

Remerciements (Acknowledgements)

I whould like to thank Jose Ismael Ripoll <iripoll@disca.upv.es> for allowing me to use his images. His RTLinux tutorial (in spanish) helps me to build this course.
`http ://bernia.disca.upv.es/~iripoll/rt-linux/rtlinux-tutorial/index.html`

I also want to thank RTLinux and RTAI developers for their help.

Références

- FSMLabs `http ://www.fsmlabs.com` ;
- RTAI `http ://www.rtai.org` ;
- Archive mailing-liste RTAI
  `http ://www.realtimelinux.org/archives/rtai/` ;
- Portail RTLinux de l'université de Valencia
  `http ://bernia.disca.upv.es/rtportal/` ;
- Comedi `http ://stm.lbl.gov/comedi/` ;
- Portail plus jeune
  `http ://www.realtimelinuxfoundation.org` ;
- Portail Linux embarqué `http ://www.linuxdevices.com` ;
- Travaux Linux embarqué / temps réel
  `http ://nferre.free.fr`.