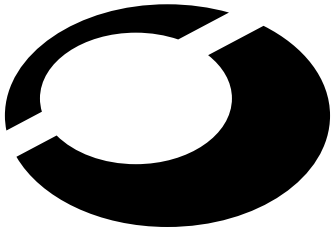
www.alcove.com

Linux Temps Réel

Nicolas Ferre
nferre@free.fr

version 1.11



l'informatique est libre

Alcôve

Copyright © Nicolas Ferre nferre@free.fr, Alcôve

Ce document peut être reproduit, distribué et/ou modifié selon les termes de la Licence GNU de Documentation Libre (*GNU Free Documentation Licence*) dans sa version 1.1 ou ultérieure telle que publiée, en anglais, par la *Free Software Foundation* ; sans partie invariante, avec comme première de couverture (*front cover texts*) les deux premières pages, et sans partie considérée comme quatrième de couverture (*back cover texts*)

Une copie de la licence est fournie en annexe et peut être consultée à l'url :
http ://www.gnu.org/copyleft/fdl.html

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

Chapitre 1 Définition et concepts 3
Chapitre 2 Linux : noyau et système 8
Chapitre 3 Amélioration du comportement temps réel 17
Chapitre 4 Approche micro-noyau 24
Chapitre 5 Conclusion 33

Chapitre 1

Définition et concepts

Les temps réel...

Définition du IEEE : *"Un système temps réel est un système dont le temps de réponse est aussi important que la qualité de fonctionnement."*

Déterminisme != vitesse : *"Le temps réel est la capacité de 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...

guaranteed worst case vs best effort

Caractérisation

Déterminisme :
- OS monotâche (DOS) ;
- multitâche **et** totalement préemptif.

Répartition du temps CPU réalisée par le moniteur système (scheduler) :
- temps-partagé ;
- FIFO ;
- Round-robin ;
- Purement prioritaire.

Caractérisation

Mesure des performances d'un système temps réel :

- Temps de réponse sur interruption / timer, Latence ;
- Régularité d'une réponse périodique : Jitter ;
- Précision d'ordonnancement des tâches.

Outils

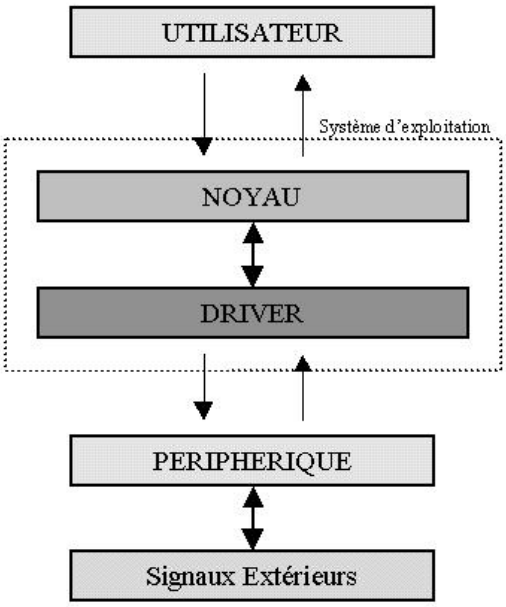
- Timers ;
- Signaux temps réel ;
- Sémaphores ;
- Queues de messages ;
- Mémoire partagée ;
- Choix de priorités et de politique d'ordonnancement.

Le noyau Linux implémente l'API "temps réel" en standard : API `POSIX.1b` (autrefois `POSIX.4`).

Linux : noyau et système

Architecture du noyau

Espace utilisateur / espace noyau.



Espace utilisateur

- Mémoire virtuelle (espace d'adressage virtuel, protections mémoire, etc.) ;
- Débogage aisé, outils classiques de recherche d'erreurs ;
- Interface graphique ;
- ...

Transmission de l'information du noyau à l'application :

- Appels systèmes (`fork(), ...`) **< 100 ms**
- `copy_from_user() / copy_to_user()`
- Interruption -> handler => processus éligible -> en cours.

Espace noyau

- API limitée au C ;
- Difficulté de débogage ;
- Modules chargeables ;
- Opérations en mode noyau non-préemptibles ;
- Masquage d'interruptions (routines `cli / sti`)

=> L'implémentation d'un comportement temps réel "dur" avec le noyau Linux n'est possible qu'au prix de sévères modifications du code du noyau.

L’ordonnanceur (scheduler)

Linux a été implémenté à partir des spécifications POSIX pour construire un système d’exploitation **temps partagé** .

=> Amélioration des performances moyennes,

=> Temps de réponse non borné.

Possibilité de mise en oeuvre de politiques "pseudo temps réel" SCHED_FIFO, SCHED_RR...

Outils : le scheduler

Modification des priorités des processus :

– `nice()` "charisme" d’un processus ;
– `sched_get_priority_max(), sched_setparam()` : priorités "temps réel" de 1 à 99.

Modification de la politique d’ordonnancement :

– `sched_setscheduler()` ;
– `sched_setparam()`.

Le calcul dynamique de priorité est fait à partir des paramètres présents dans la structure `task_struct : policy, priority, rt_priority, counter`.

Outils : IPC

IPC système V :

- Tubes (tubes anonymes `pipe()`, `popen()` / nommés `mkfifo()`) ;
- Files de messages (`msgget()`, `...`) ;
- Sémaphores (`semget()`, `...`) ;
- Mémoires partagées (`shmget()`, `...`).

Performances, utilisation

- Durée maximale d'un appel système : 100 ms ;
- Résolution d'un timer (fixée par la constante `CLOCK_TICK_RATE`) : 10 ms ;
- Latence d'un handler d'IT noyau sur x86 : < 15 micro secondes.

Références

- Kernel Hackers Guide
  http ://www.linuxdoc.org/LDP/khg/HyperNews/get/khg.html
- The Linux Kernel http ://www.linuxdoc.org/LDP/tlk/tlk.html
- The Linux Kernel Module Programming Guide
  http ://www.linuxdoc.org/LDP/lkmpg/mpg.html
- Understanding the Linux Kernel (scheduler)
  http ://www.oreilly.com/catalog/linuxkernel/chapter/ch10.html

Amélioration du comportement temps réel

Scheduler à priorités fixes

Version beta par *MontaVista* .

Remplacement du scheduler dans le noyau `sched.c -> rtsched.c`.

- Modification de la queue d'éligibilité (Ready Queue).
- 127 niveaux de priorité "temps réel" ;
- Séparation des tâches temps réel des autres ;
- => **indépendance vis-à-vis de la charge** ;
- => **rapidité de changement de contexte** .

Réduction de la latence

Réduction de la latence des 10 ms typiques à une valeur <5 ms.

Patch *LowLatency* conçu pour améliorer les performances multimédia du noyau Linux.

Amélioration de l'algorithme de préemption du noyau :

- Découpage de tout délai engendré par un appel système, un accès aux périphériques, pour réordonnancer les processus ;
- Réduction des zones critiques ;
- Appel au scheduler avant toute action de longue durée, dans le code du noyau.

Intallation

Simple patch au noyau officiel :

- Téléchargement du noyau officiel correspondant au patch ;
- Déarchivage du noyau et copie du patch dans le répertoire `/usr/src` ;
- Liaison symbolique : `ln -s linux-2.x.x linux` ;
- Patch du noyau : `patch -p1 < ../patch-xx` ;
- Configuration du noyau : `make menuconfig (| xconfig)` ;
- Compilation du noyau : `make dep clean bzImage modules modules_install` ;
- Copie du noyau, configuration de *lilo* (`/etc/lilo.conf`) ;
- Lancement de *lilo* : `/sbin/lilo` et reboot de la machine.

Critères de choix

- Avantages :
  - Programmation avec les outils classiques, en espace utilisateur ;
  - Conception traditionnelle ;
  - Pas de modification des drivers ;
  - Simplicité d'installation.
- Inconvénients :
  - Pas de possibilité d'offrir un comportement temps réel **critique** ;
  - Difficulté d'écriture du patch (noyau Linux = 40 Mo de code) audit de chaque action du noyau => obsolescence ;
  - Difficulté de maintenance du patch et introduction de bugs.

Références

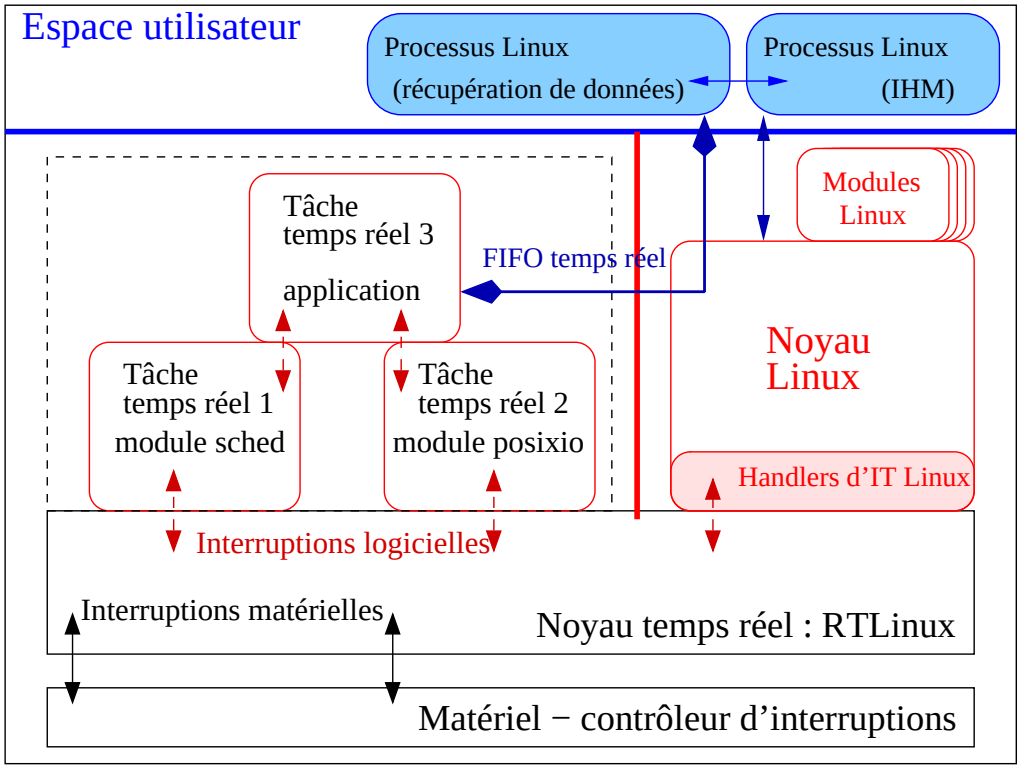
- MontaVista http ://www.mvista.com
- Scheduler temps réel ftp ://ftp.mvista.com/pub/Real-Time/
- Interrupt Latency http ://www.mvista.com/realtime/latency/
- Lowlatency patch http ://www.gardena.net/benno/linux/audio/
- TimeSys http ://www.timesys.com
- Comparaison de deux approches du temps réel sous Linux http ://www.linuxdevices.com/articles/AT7005360270.html

Approche micro-noyau

Architecture

- Les tâches
  - Linux est une tâche temps réel de faible priorité
  - Dialogue entre tâches temps réel et processus Linux par FIFOs ou mémoire partagée
  - Les FIFOs permettent d'éviter l'inversion de priorité
- Les interruptions
  - Le code de masquage et démasquage des interruptions a été réécrit
  - Les interruptions « hard » sont déroutées par la couche temps réel
  - Générations d'interruptions « soft » à Linux
  - Linux ne peut intervenir sur les interruptions « hard »

Architecture



Implémentation

- Les tâches RTLinux se présentent sous forme de modules noyau (chargeables dynamiquement)
- Allocation statique de toutes les ressources
- Communications non-bloquantes côté temps réel

Outils

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

- `rtl_sched` : scheduler exclusivement prioritaire ;
- `rtl_time` : horloge processeur et services timers, routine de traitement d'interruptions... ;
- `rtl_posixio` : API de style POSIX (read/write/open...) pour interfacer les drivers ;
- `rtl_fifo` : communication avec Linux ;
- ...

=>Système modulaire et extensible.

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."

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

Communications non-bloquantes côté temps réel.

Utilisation de modules chargeables / API spécifique.

Intallation

- Récupération du noyau standard Linux et du patch temps réel
- Modification et configuration du noyau
- Compilation du noyau
- Boot et test

Critères de choix

- Avantages :
  - Comportement temps réel critique ;
  - Modularité, scalabilité ;
  - Compacité du code et du patch noyau (64 ko/qq 100 lignes) :
    - Version récente du noyau Linux (dernières innovations) ;
    - Peu de bugs, correction rapide, testé par de nombreux spécialistes ;
- Inconvénients :
  - Modification du code des drivers pour un comportement temps réel ;
  - Apprentissage d'une API (facilité par le module POSIX) ;
  - Difficulté de débogage (espace noyau).

Références

- FSMLabs http ://www.fsmlabs.com
- RTAI http ://www.rtai.org
- Portail communautaire général http ://www.realtimelinux.org
- Portail plus jeune http ://www.realtimelinuxfoundation.org
- Page performances Lineo http ://pets.lineo.com/index.php
- Portail Linux embarqué http ://www.linuxdevices.com
- Travaux Linux embarqué / temps réel http ://nferre.free.fr

Conclusion

Choix

- Necessité d'évaluer le besoin de l'applicatif ;
- Evaluer la maturité / pérénité de l'environnement ;
- Grandeurs caractéristiques obtenues par des benchmarks objectifs.

Perspectives

- Intégration dans les versions futures du noyau (multimédia, streaming, etc.) ;
- Développement plus aisé en espace noyau (module de débogage, interaction avec l'espace utilisateur) ;
- Richesse et diversité des différents projets libres ;
- Linux dans le monde de l'embarqué => monde du temps réel.