Formation à Linux Embarqué

Jérôme Pouiller <j.pouiller@sysmic.org>

Cinquième partie V

Linux et le temps réel

22 Systèmes Symétriques et Asymétriques

23 Problématique des OS RT

24 Bases

25 Latences

26 Low latency

27 Hyperviseurs

28 RT-préempt

Architectures multi-coeurs

Il est possible d'utiliser des architecture multi-coeurs pour des systèmes temps réels.
On distingue alors 2 types d'architectures :

- Les systèmes symétriques (SMP) où les tâches ne sont pas affectées à un CPU particulier
- Les systèmes asymétriques où on affecte manuellement les tâches à un CPU

Systèmes asymétriques

Les systèmes asymétriques sont assez simples à architecturer :

- On effectue une étude séparées pour chaque CPU
- On considère les échanges entre les CPU comme des entrées/sorties

Sous Linux, il est possible d'associer une tâche avec un CPU grâce à :

- la fonction *CPU affinity*
- la commande `taskset` ou `lxc`
- aux fonctions *cgroup* et *cpuset*

Systèmes symétriques

- Beaucoup plus complexe à architecturer
- La plupart de nos algorithme ne sont plus prouvé dans une architecture SMP
- La technologie est relativement récente (fin des années 90)
- Pourquoi si tard ?
  - Principalement problèmes matériels
  - Problèmes de barrières mémoires (hard ou soft)
  - Algorithme beaucoup plus ardu (inversion de priorité par exemple)

Limites des système classiques

Les systèmes classiques s'appuient sur un système d'exploitation en général mal adapté pour le temps réel :

- Politique d'ordonnancement visant à équilibrer équitablement le temps alloué à chaque tâche
- La gestion de la mémoire virtuelle, des caches engendrent des fluctuations temporelles
- La gestion des temporisateurs qui servent à la manipulation des temps pas assez fin
- Mécanismes d'accès aux ressources partagées et de synchronisation comportent des incertitudes temporelles
- Gestion des interruptions non-optimisées
- Systèmes non-certifiés
- API mal adaptées au systèmes temps réels

sysmic

Mécanisme basiques

- Possibilité d'exécuter une tache avec `SCHED_RT` ou `SCHED_FIFO` (priorité toujours inférieure aux *sofirqs* du noyau)
- Possibilité de marquer les pages de mémoire avec `mlock`
- L'accès à des horloges haute précision est un élément clef d'un système temps réel.
  - Le framework *hpet* (*High Precision Timers*) est assez récent dans le noyau Linux (2001) (à titre de comparaison, l'équivalent de *hpet* est apparu sous Windows à partir de Vista).
  - Autrefois, l'horloge était utilisée pour l'ordonnanceur. Le temps retourné par les différents services était celui calculé par l'ordonnanceur. Avec *hpet*, les horloges sont des périphériques à part entière

LXC

`lxc` permet de créer des container et d'avoir une gestion fine des limitations de ressources.

- Activer *Control Group Support* dans le noyau
- Démonstration avec `lxc-test.c`

```
$ gcc samples/lxc-test.c -o lxc-test
$ apt-get install lxc
```

- Création d'un container et éxecution de la commande

```
$ lxc-create -n test
$ lxc-execute -n test ./lxc-test
```

LXC

- Limiter le container à au CPU 0 :

```
$ lxc-cgroup -n test cpuset.cpus 0
```

- Supprimer la limite :

```
$ lxc-cgroup -n test cpuset.cpus 0,1
```

- Limiter la consommation cpu à 10% :

```
$ lxc-cgroup -n test cpu.cfs_quota_us 10000
```

- Retirer la limite :

```
$ lxc-cgroup -n test cpu.cfs_quota_us -1
```

- Connaitre la consommation CPU du container :

```
$ lxc-cgroup -n test cpuacct.usage
```

- cf. `/sys/fs/cgroup/`

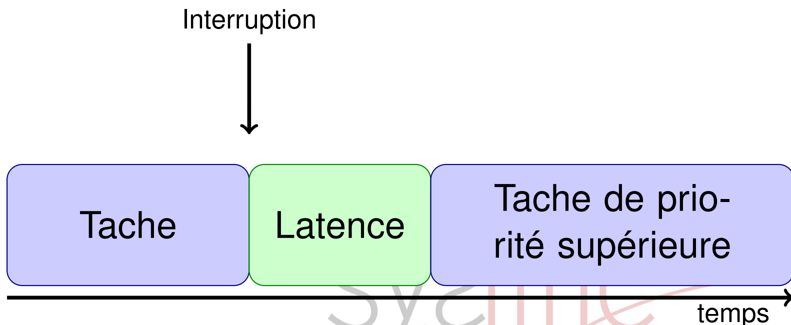
Temps réel dans le noyau

Travailler dans le noyau :

- Possibilité de travailler sous interruptions
- ... ou de travailler avec les soft IRQ et/ou les tasklets
- Oblige à travailler dans le noyau (problèmes d'APIs, etc...)
- Nécessaite beauoup de connaissance pour faire quelque chose de bien
- Accès difficiles aux bibliothèques de haut niveau (un simple accès au réseau peut devenir très complexe)
- Gardez le kernel pour faire des drivers, externalisez le traitement

Latence aux évènements

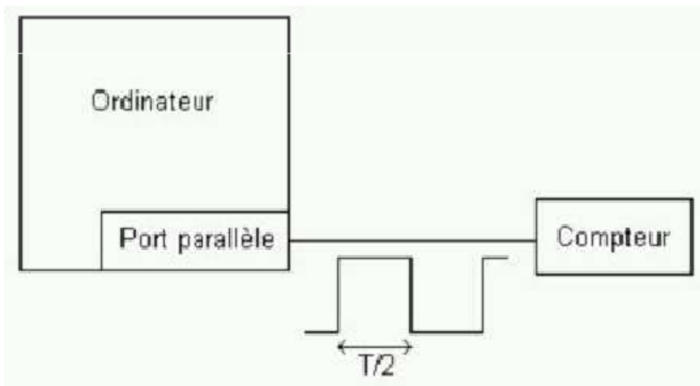
- Coeur du problème
- Si la latence était nul (ou au moins constante), on calculerait simplement le temps de réponses de nos tâches.



Latence aux évènements

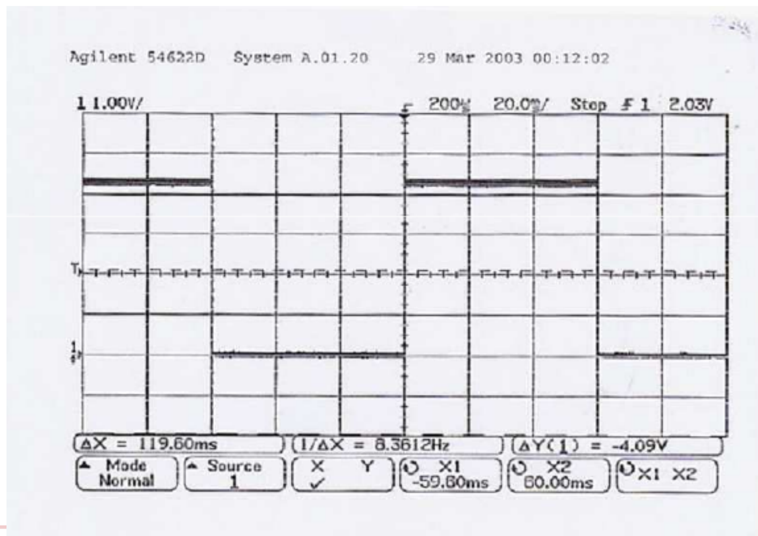
Exemple concret

- On paramètre un timer à 50Hz
- On mesure le temps effectif de chaque période



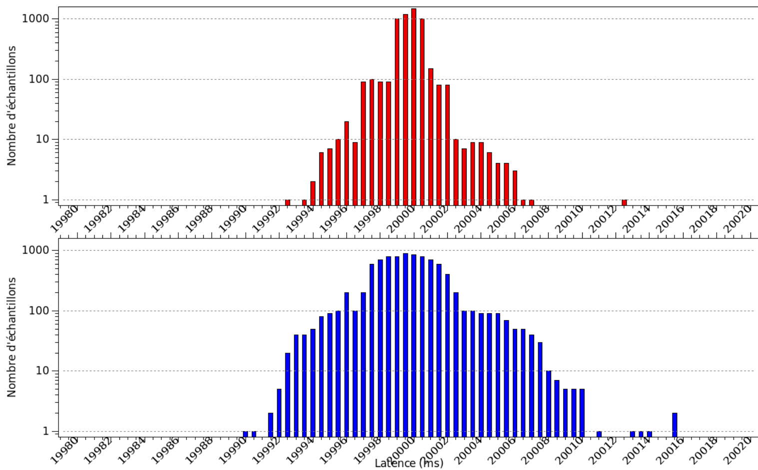
Latence aux évènements

Système temps réel



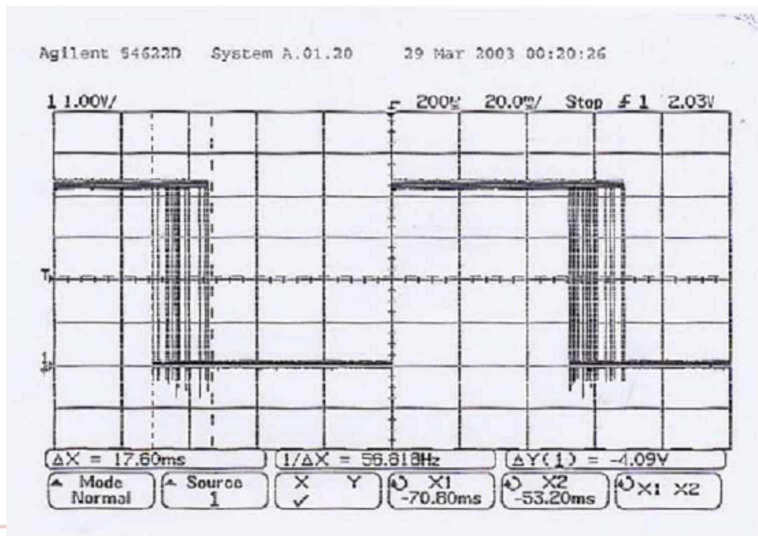
Latence aux évènements

Système temps réels



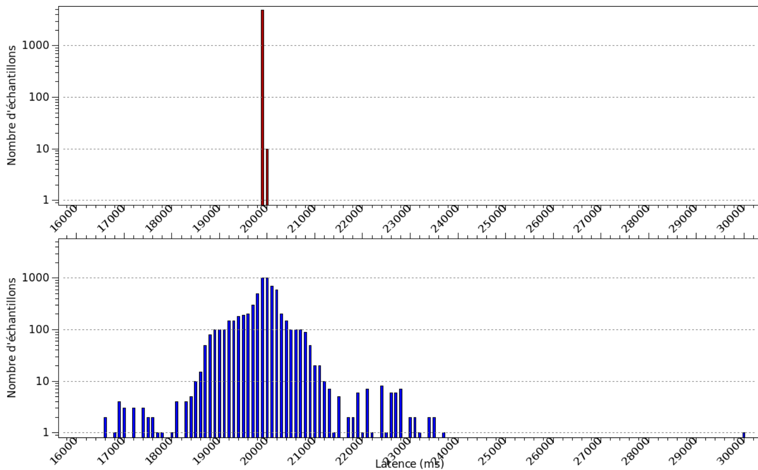
Latence aux évènements

Système classique



Latence aux évènements

Système classique



Contexte

Ordonnancement statique :

- Charge des taches temps réelles $\ll$ 100%

$\rightarrow$ Ordonnancement avec des priorités statiques suffisante

$\rightarrow$ Problématique des algorithmes d'ordonnancement à priorité dynamique (EDF, LST, etc...) secondaire

Noyau low latency

Problème du noyau normal

Dans un noyau Linux classique, il y a un seul contexte noyau pour tout le monde

- Pas possible de préempter le système
- Personne ne peut prendre la main lorsqu'un processus est dans un *syscall* (ordonnanceur désactivé)
- Équivalent d'une ressource partagée par tout les processus

→ Latence

Noyau low latency

Implémentation

- Difficile de gérer les différents contextes noyau

→ Noyau réentrant (= thread-safe)

- Gestion des interruption assez complexe
- Overhead assez important

Noyau low latency

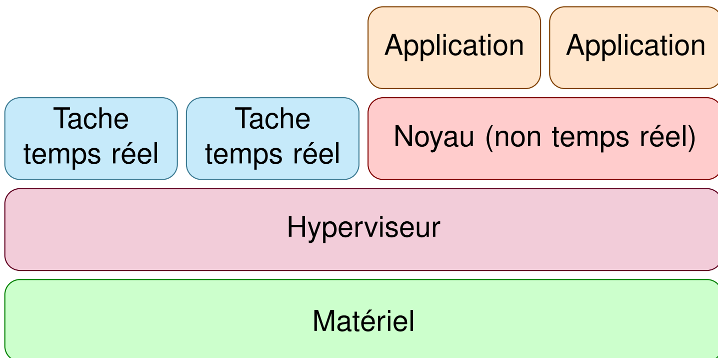
Résultats

- Patch low-latency mergé dans le mainstream avec le noyau 2.6 (`CONFIG_PREEMPT`)
- Latences maximum de l'ordre de 300μs (chiffres de l'époque)

Hyperviseur

- Noyau low latency encore problématique pour les interruptions
- Beaucoup d'interruptions $\rightarrow$ latence
- Hyperviseur

Temps réel
Nano Kernel



Hyperviseur

- Possibilité de préempter le noyau sans patch low-latency
- Possibilité de différer les interruptions
- Possibilité d'ignorer les interruptions dans les sections temps réelles
- Performances excellentes ($< 20\mu s$)
- Technique non spécifique à Linux
- Peu de code en mode temps réel

→ Certifiable

- Fonctionne au dessus du noyau
- Comportement des interruptions à modifier
- Communication entre les tâches temps réelles et le reste

→ Nécessite de patcher le noyau

- RTLinux, RTAI et Adeos (http://download.gna.org/adeos/patches/, git://git.xenomai.org/ipipe-gch.git) (Xenomai)

Hyperviseur

Adeos

- Fork de RTAI
- API utilisateur assurée par Xenomai
  - Skins
  - → Possibilité de faire fonctionner une application développée pour vxWorks
  - Skin native consistante
  - Beaucoup mieux que Posix (de plus, il existe une skin Posix)
  - http://www.xenomai.org/documentation/xenomai-head/html/api

Adeos

Installation du patch noyau

```
$ wget https://xenomai.org/downloads/ipipe/v3.x/arm
    /older/ipipe-core-3.10.32-arm-9.patch
$ wget http://download.gna.org/xenomai/stable/
    xenomai-2.6.4.tar.bz2
$ tar xvf xenomai-2.6.4.tar.bz2
$ xenomai-2.6.4/scripts/prepare-kernel.sh --ipipe=
    ipipe-core-3.10.32-arm-9.patch --arch=arm --
    linux=linux-3.10.32
$ cd linux-3.10.32
$ make ARCH=arm menuconfig
$ make ARCH=arm uImage modules
$ sudo make INSTALL_MOD_PATH=/home/user/nfs ARCH=
    arm modules_install
$ sudo mknod /home/user/nfs/dev/rtheap c 10 254
$ sudo mknod /home/user/nfs/dev/rtscope c 10 253
$ sudo mknod /home/user/nfs/dev/rtp c 150 0
```

10

Adeos

Installation (Autotools classiques) de la bibliothèque Xenomai :

```
$ cd ../xenomai-2.6.4
$ ./configure --host=arm-linux --enable-shared=yes
    --enable-smp
$ make
$ sudo -E make DESTDIR=/home/user/nfs install
```

- Possibilité de faire la même chose avec Buildroot
- Il sera ensuite nécessaire de compiler les programme Xenomai avec le résultat de

```
$ xeno-config --skin=native --ldflags/--cflags
    | sed 's:=/:=/home/user/nfs/'
```

Xenomai

Les choses intéressantes :

- Charger le système :

```
target# mount -t jffs2 /dev/mtdblock1 /mnt
target# /usr/xenomai/bin/dohell -m /mnt 10
```

- Vérifier la latence d'une tâche périodique :

```
target# /usr/xenomai/bin/latency -t 0
```

  Testons ensuite avec `-t 1` et `-t 2`

- Vérifier le temps de calcul sur des flottants :

```
target# /usr/xenomai/bin/arith
```

  Les valeurs *rejected* ont eu un temps de calcul > à 4 fois la moyenne (même si je ne suis pas tout à fait d'accord avec leur manière de compter).

RT-preempt

Gestion des interruption dans des threads

- Permet de préempter les interruptions

→ Moins de latence des taches

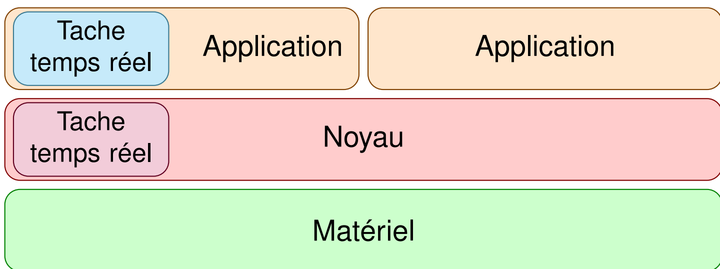
- Permet d'ordonnancer les interruption

→ Permet de se passer de la désactivation des interruptions

→ Permet de remplacer les spin lock par des mutex

→ Moins de latence des interruptions

RT-preempt



RT-preempt

- Patch RT (Patchs : http://www.kernel.org/pub/linux/kernel/projects/rt/, Git : git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/linux-rt.git)
- Implémentation assez complexe

→ Non portable

- Performance très bonnes (20μs)
- Intégré en partie dans le noyau 3.0