Panorama des temps réels sous GNU/Linux

Open Wide
Architecte Open Source

Stelian Pop <stelian.pop@openwide.fr>

Le besoin

- en contexte industriel il y a souvent besoin d'un comportement temps réel:
  - pilotage matériel (moteurs, ...)
  - acquisition de données (capteurs, ...)
  - piles protocolaires (*GPRS*, *VoIP*, ...)
  - etc.
- quelques fois, la contrainte temps réel est induite par le design de l'applicatif (exemple: portage de code depuis un autre OS temps réel)

Les solutions

- noyau Linux standard
- noyau Linux préemptif
- noyau Linux PREEMPT_RT
- *RTLinux*
- *RTAI*
- *Xenomai*
- ... et d'autres solutions plus confidentielles (*Jaluna C5*,...)

Linux standard

- solution clé en main (distributions généralistes ou spécialisées)
- disponible sur 24 architectures (et beaucoup de plate-formes) !
- très bonnes performances globales
- stable, mature (15 ans...)

Linux standard fonctionnement (1/2)

- (presque) tout le traitement s'effectue dans des processus
- les processus s'exécutent en mode utilisateur (l'algorithmique) ou en mode noyau (les entrées-sorties)
- les gestionnaires d'interruptions et exceptions s'exécutent hors processus, préemptent tout traitement, puis déclenchent des traitements dans des processus (en mode noyau)

Linux standard fonctionnement (2/2)

- il est possible de désactiver les interruptions (et beaucoup de pilotes le font...)
- un processus en mode noyau n'est pas préemptible par un autre processus
- un processus en mode utilisateur est préemptible par l'ordonnanceur:
  - l'ordonnanceur utilise l'interruption *timer*
  - la cadence du *timer* est fixe (*HZ* = 1-10 ms)
- on peut gérer la priorité entre les processus (au niveau de l'ordonnanceur)

Linux standard utilisation et performances

- écriture de code noyau:
  - API spécifique, limitée aux opérations "noyau"
  - langage C
  - temps de latence moyen: dépendant du contexte
- écriture de code applicatif
  - multiples API fournies par des bibliothèques
  - multiples langages (C, C++, Python etc)
  - temps de latence moyen: constante *HZ*
- n'est pas temps réel (mais *good enough...)*
- temps de latence maximum important, non borné, dépendant des pilotes utilisés

Linux préemptif

- solution issue des travaux de *Montavista (Robert Love),* disponible en tant que patches pour les noyaux 2.4 et intégrée dans les noyaux 2.6
- extension de l'ordonnanceur standard de Linux
- disponible sur l'ensemble des plate-formes
- améliore *l'interactivité* du système au détriment des performances globales
- stable (mais pas sur toutes les plate-formes...)

Linux préemptif fonctionnement

- un processus en mode noyau peut être préempté par un autre processus
- mais la prise d'un verrou noyau (*spinlock*) désactive la préemption
- deux variantes:
  - `PREEMPT_VOLUNTARY`: rajout de quelques nouveaux points de préemption dans le noyau (en réutilisant `might_sleep()` prévu initialement pour le déboggage des sémaphores)
  - `PREEMPT_DESKTOP`: préemption possible à chaque retour d'interruption ou exception

Linux préemptif utilisation et performances

- même utilisation que Linux standard:
  - écriture de code noyau
  - écriture de code applicatif
- le coût de la préemption n'est pas nul, les performances globales du système diminuent
- temps réel soft
- pas d'incidence sur le temps de latence moyen
- temps de latence maximum toujours important, mais avec moins d'amplitude

Linux PREEMPT_RT

- nouveau développement pour le noyau 2.6, essentiellement mené par *Ingo Molnar*: people.redhat.com/mingo/realtime-preempt
- extension de l'ordonnanceur standard de Linux (respect strict des priorités et prévention des PI)
- disponible sur l'ensemble des plate-formes
- en développement, testé essentiellement sur *x86*
- implications encore mal comprises...
- nécessite que tous le code du noyau (+ pilotes) soit revu (et adapté...)

Linux PREEMPT_RT fonctionnement

- la prise d'un verrou n'empêche plus la préemption (les *spinlock* deviennent des *mutex*)
- les interruptions sont "*threadées*" (les handlers d'interruption s'exécutent en contexte processus en mode noyau et sont préemptibles)
- afin d'éviter le blocage *(deadlock)* par inversion de priorité sur un *mutex*, un mécanisme d'héritage de priorité est utilisé

Linux PREEMPT_RT utilisation et performances

- utilisation: identique à Linux standard, mais avec la possibilité de:
  - choisir pour chaque interruption de *threader* ou pas
  - choisir la priorité de la thread correspondante
- le coût de la préemption peut être important
- temps réel dur
- temps de latence moyen de l'ordre de 20-30 microsecondes
- temps de latence maximum de l'ordre de 100-500 microsecondes (largement dépendant de la configuration matérielle et logicielle)

RTLinux

- la plus connue solution Linux temps réel
- initialement développé en tant que thèse universitaire par *Michael Barabanov* (avec *Victor Yodaiken)* au *New Mexico Tech (NMT)* en 1997
- devenu propriétaire en 2000 suite à la création de *FSMLabs*
- deux versions:
  - version GPL www.rtlinux-gpl.org
    - *i386* (stable), *alpha, mips, powerpc*
    - en retard, abandonnée...
  - version propriétaire www.fsmlabs.com
    - supporte la plupart des plate-formes
    - stable, mature

RTLinux fonctionnement

- technique de *co-noyau*, séparation entre le comportement temps réel (*RTLinux*) et le comportement non temps réel (Linux)
- RTLinux est un petit noyau temps réel complètement préemptible
- les interruptions sont interceptées par *RTLinux* et propagées à Linux
- *RTLinux* exécute en priorité les tâches temps réel et exécute le noyau Linux en tant que tâche de faible priorité (boucle *idle*)
- le code du noyau Linux est modifié pour ne pas toucher aux interruptions matérielles (virtualisation)

RTLinux
utilisation et performances

- utilisation:
  - API de type threads *POSIX* pour les besoins temps réel (+pilotes), mode noyau uniquement
  - développement Linux standard (noyau et applicatif) pour les besoins non temps réel
  - communication entre les tâches *RTLinux* et les processus Linux par des *FIFO* ou mémoire partagée
- temps réel dur
- temps de latence maximum de l'ordre de 10 microsecondes

RTAI

- adaptation des premières versions de RTLinux (1999) par le *Département d'Ingénierie Aérospatiale de l'école Polytechnique de Milan*
- suite à la création de *FSMLabs*, RTAI devient un projet à part, évoluant indépendamment
- supporte *i386* (stable), *arm, cris, mips, powerpc*
- stable, mais évolutions chaotiques

RTAI – fonctionnement

- même technique que *RTLinux* (*co-noyau*)
- les versions actuelles de *RTAI* utilisent la couche *Adeos* pour faire du *pipelining* d'interruptions (à la place de la *vectorisation* directe)
  - meilleure architecture logicielle
  - défense contre un brevet logiciel de *FSMLabs*

    ... mais Adeos est lui même contourné dans les toutes dernières versions de RTAI (3.2)
- possible de migrer des tâches entre l'ordonnanceur Linux et l'ordonnanceur *RTAI* (*LXRT, fusion*)
- émulation d'autres APIs (*POSIX, VxWorks* etc)

RTAI - performances

- utilisation:
  - API spécifique pour les besoins temps réel (+pilotes), mode noyau ou *LXRT*
  - développement Linux standard (noyau et applicatif) pour les besoins non temps réel temps réel dur
  - communication entre les tâches *RTAI* et les processus Linux par des *FIFO* ou mémoire partagée
- temps réel dur
- temps de latence maximum de l'ordre de 10 microsecondes

Xenomai

- projet de "OS temps réel abstrait", émulant les autres API par un mécanisme de personnalités (*skin*) développé par *Philippe Gerum* (2000)
- intégré à *RTAI* en 2004 (branche *RTAI/fusion*), devait servir de coeur pour la futur *RTAI*
- suite à des divergences entre développeurs, *Xenomai* quitte *RTAI* et devient un OS temps réel à part entière (2005)
- plate-formes supportées: *arm, blackfin, i386, ia64, powerpc32, powerpc64*
- jeune mais stable

Xenomai fonctionnement (1/2)

- technique de *co-scheduler* intégré dans le noyau
- *Xenomai* utilise les services de *Adeos* pour la virtualisation des interruptions (même auteur...)
- *Xenomai* est composé de:
  - *nucleus* + mécanismes de communication (OS temps réel abstrait)
  - diverses API implémentant les *skins*: *natif, posix, rtai, psos, vxworks*, etc.
- mécanisme automatique de migration des tâches entre l'ordonnanceur Linux et *Xenomai* en fonction des API utilisées (mode secondaire/primaire)

Xenomai
utilisation et performances

- utilisation:
  - API des skins temps réel pour les besoins temps réel, mode noyau ou utilisateur (API commune)
  - développement Linux standard (noyau et applicatif) pour les besoins non temps réel
  - communication entre les deux par différents moyens, en fonction du skin utilisé
- temps réel dur
- temps de latence de l'ordre de 10-15 microsecondes

Résumé
comparaison des solutions

	Linux	Linux préemptif	Linux PREEMPT_RT	RTLinux	RTAI	Xenomaï
Type	scheduler standard	optimisation scheduler	optimisation scheduler	co-noyau	co-noyau	co-scheduler
Plate-forme	toutes	toutes (problèmes possibles)	toutes (problèmes possibles)	***i386***, *alpha, mips, ppc (+propriétaire)*	***i386***, *ppc, arm, mips, cris*	***arm, blackfin, i386, ia64,***
API	noyau ou toutes API user	noyau ou toutes API user	noyau ou toutes API user	POSIX noyau	RTAI noyau ou LXRT user	multiples API communes noyau/user
Temps de latence moyen	HZ (1-10 ms)	HZ(1-10 ms)	20 – 30 us	5 us	5 us	7 us
Temps de latence max	∞	∞	100-500 us	10 us	10 us	10 – 15 us
Etat	stable	stable	en développement	stable (+propriétaire)	stable, évolutions chaotiques	stable, jeune

Fin...

- Solutions:
  - Linux, Linux préemptif: http://www.kernel.org
  - PREEMPT_RT: http://people.redhat.com/mingo/realtime-preempt
  - *RTLinux*: http://www.fsmlabs.com
  - *RTLinux GPL*: http://www.rtlinux-gpl.org
  - *RTAI*: http://www.rtai.org
  - *Xenomai*: http://www.xenomai.org
- Transparents disponibles à:
  - http://www.popies.net/conferences/tempsreels.pdf