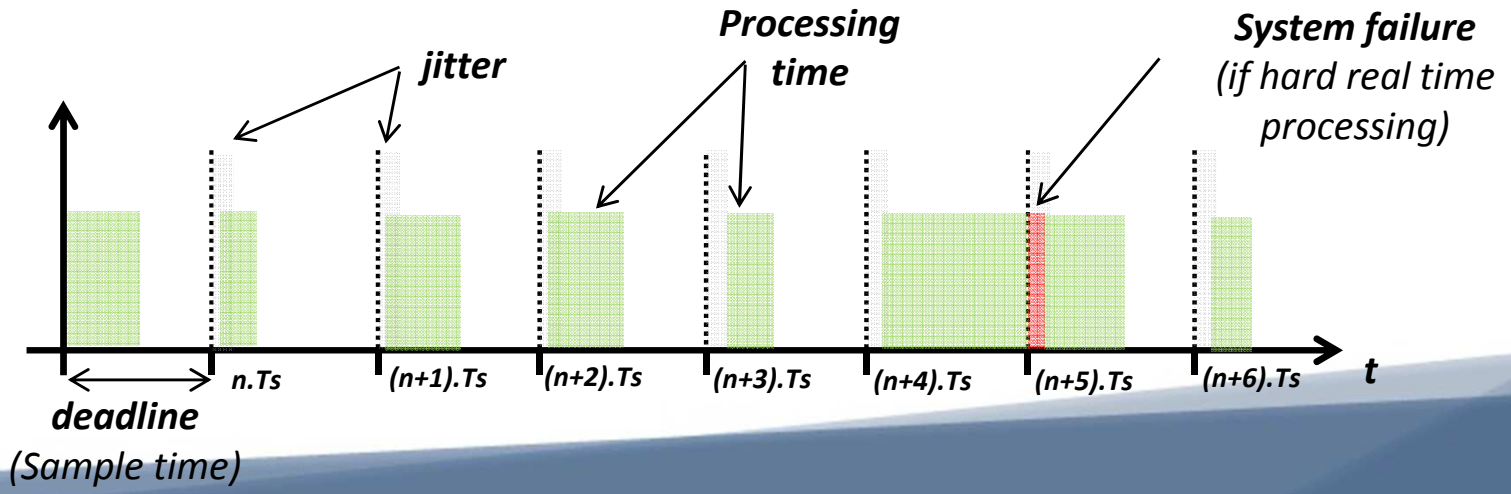
ENSICAEN
Embedded Systems

REAL TIME LINUX

Embedded Linux

Hugo Descoubes – mars 2014

Un système temps réel doit satisfaire à des contraintes temporelles et garantir un déterminisme logique et temporel à l'exécution. *Attention, temps réel ne veut pas forcément dire rapide (régulation de température, régulation de procédés chimiques, applications climatiques …). Une solution technologique se voulant temps réel est le plus souvent la résultante d'une alchimie entre choix de la solution matérielle (MCU, FPGA, SoC, GPP … cache-less, out-of-order-engine CPU …) et logicielle portée (OS-less, OS, RTOS) :*



Rappelons les différentes hiérarchies et terminologies de temps réel cohabitant souvent dans une même application :

- ***Temps réel dur (Hard Real Time) :*** *échéance temporelle vitale/critique à garantir (assistance au freinage, guidage de missile, pacemaker …)*

- ***Temps réel ferme (Firm Real Time) :*** *échéance temporelle à garantir mais non vitale, résultat inutile si deadline dépassée mais applicatif toujours viable (vidéo, audio …)*

- ***Temps réel mou (Soft Real Time) :*** *échéance à garantir mais réponses tardives et latences tolérées (IHM, connectivités …)*

- ***Non temps réel :*** *échéance non garantie (debug, profilage …)*

Rappelons que dans le cadre de développement d'applications pouvant être temps réel, plusieurs solutions s'offrent au développeur :

- ***Séquenceur (homemade) :*** *Système OS-less sans OS/RTOS avec fonction ''main'' implémentant un séquenceur cyclique (exemple de l'offline scheduling). Solutions peut évolutives et pouvant être difficiles à maintenir.*

- ***Real Time Operating System (exécutif temps réel) :*** *exécutifs proposant le plus souvent très peut de services (scheduling, parfois stack IP, file system … souvent sans gestion de MMU et drivers non inclus) mais offrant une empreinte mémoire faible (on-chip memory) ainsi qu'un grand déterminisme temporel avec très peu de jitter. Solutions évolutives et permettant un coût matériel global réduit. Exemple :* ***FreeRTOS, RTX, VxWorks, uC/OS-III, SYS/BIOS …***

- ***Standard Operating System :*** *les systèmes d'exploitation dans le monde du PC (famille Windows, MAC OS X, distro GNU/Linux …) sont des systèmes temps réel mou. Toute application lancée aura, tôt ou tard, sa chance afin d'accéder à une ressource CPU, ceci est du à la politique d'ordonnancement (scheduling policy) de ces systèmes et à la nature non vitale/critique des applicatifs.*

Dans ce chapitre, nous allons nous intéresser à rendre un système GNU/Linux temps réel puis nous validerons les performances de notre système à l'aide d'outils de benchmarking.

Observons les principaux acteurs du marché proposant une évolution vers une solutions Linux temps réel :

- ***Xenomai*** *(http://www.xenomai.org/)* ***:*** *solution proposant un co-noyau temps réel (nucleus) cohabitant avec le noyau standard Linux. Xenomai est porté sur un hyperviseur (Adeos ou Adaptative Domain Environment for Operating Systems) ou nanokernel HAL (Hardware Abstraction Layer). Cet hyperviseur assure notamment le routage des interruptions des couches matérielles vers les systèmes ou domaines portés (Interrupt Pipe). Plusieurs API temps réel ou skins sont accessibles. Néanmoins, des skins maison peuvent être développées même si POSIX reste à privilégier pour des soucis de portabilité. En 2003, Xenomai fusionne avec le projet RTAI pour devenir un projet indépendant en 2005.*

- ***Linux-rt ou patch PREEMPT-RT*** *(https://rt.wiki.kernel.org)* ***:*** *branche du kernel proposant un patch pour le noyau linux standard permettant de le rendre temps réel. Le patch PREEMPT-RT influe sur la politique d'ordonnancement en rendant notamment préemptible la majeure du code du noyau et mettant par exemple en œuvre des mécanisme de d'héritage de priorité pour la gestion des sémaphores et des spinlocks (API kernel).*

- ***RTAI (Real Time Application Interface) :*** *extension temps réel du noyau travaillant également avec l'hyperviseur Adeos (cf. Xenomai). Il s'agit d'un projet universitaire initialement conçu comme une variante de RTLinux (projet maintenu puis abandonné en 2011 par Wind River).*

Avant d'installer nos premiers systèmes Linux temps réel, validons les outils de test sur un système standard. Nous utiliserons les utilitaires ***rt-tests*** *et notamment l'outil* ***Cyclictest*** *(cf. https://rt.wiki.kernel.org/index.php/Cyclictest). Récupérer les sources des utilitaires, les cross-compiler et installer l'utilitaire Cyclictest dans le rootfs de la BBB :*

Côté host

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git
cd rt-tests
export PATH=/<cross_compiler_path>/bin/:$PATH
export CC=arm-linux-gnueabihf-gcc
make CROSS_COMPILE=arm-linux-gnueabihf- all
cp ./cyclictest /media/rootfs/usr/bin/
```

Côté BBB

```
cyclictest –help
cyclictest -t1 -p 80 -n -i 10000 -l 10000
```

Observons en pseudo code le travail de stress réalisé par l'utilitaire Cyclictest et analysons les informations de sortie :

```
clock_gettime((&now))
next = now + interval

while (!shutdown) {
        clock_nanosleep((&next))
        clock_gettime((&now))
        diff = calcdiff(now, next)
        # update stat-> min, max, avg, now, cycles
        # Update the histogram
        next = now + interval
}
```

```
# cyclictest -t1 -p 80 -n -i 10000 -l 10000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.27 0.06 0.02 1/55 232

T: 0 (  232) P:80 I:10000 C:    297 Min:     14 Act:   18 Avg:   21 Max:   Max:      40
```

Process priority 80

10000us interval (deadline)

10000 repetitions

All numbers in uS

Process and cyclictest informations

Min latency

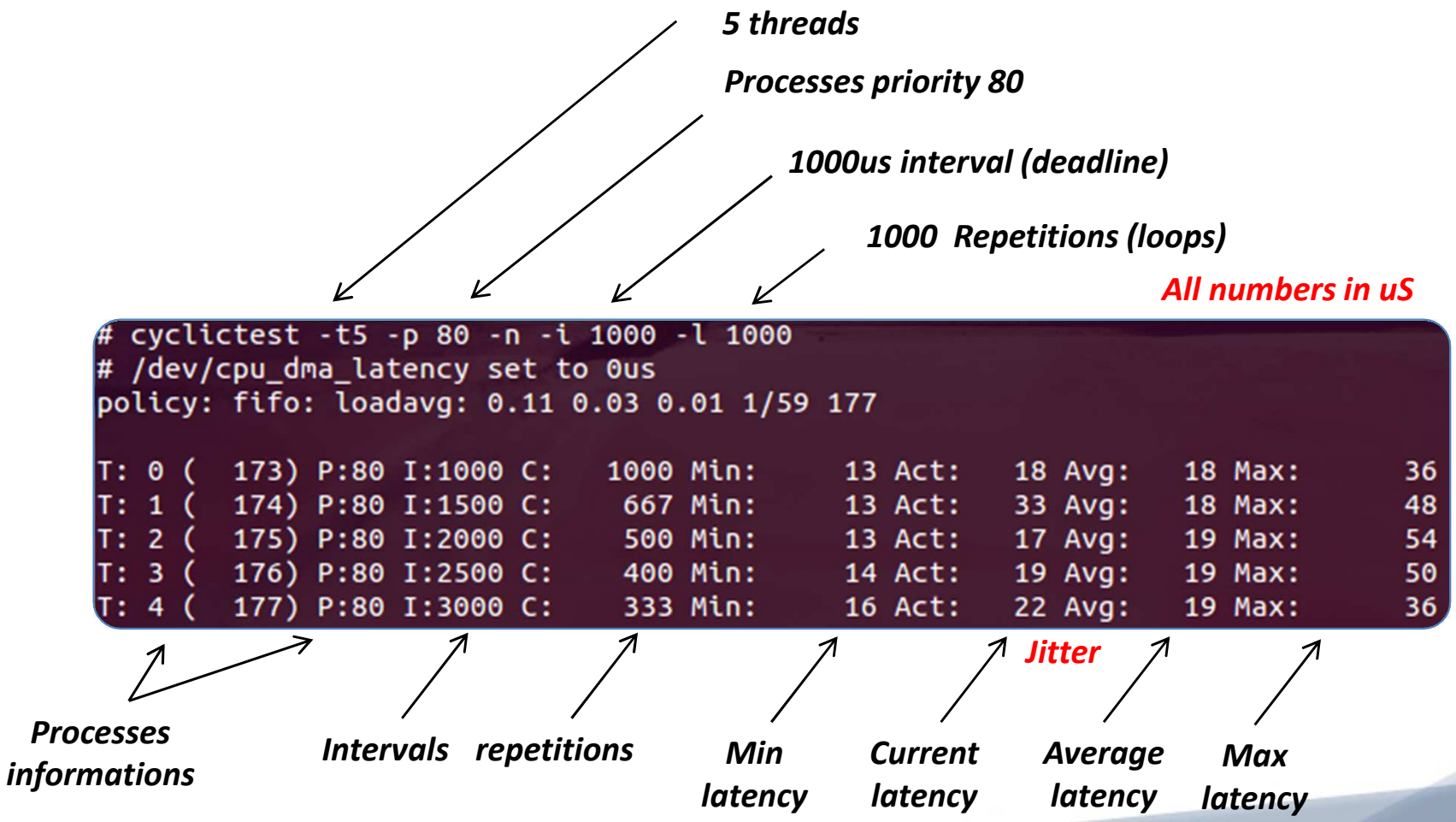
Current latency

Jitter

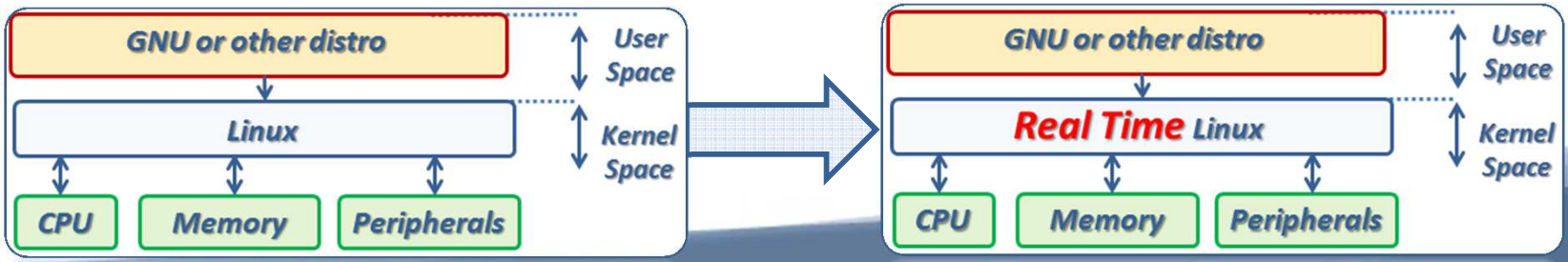
Average latency

Max latency

Prenons un exemple de test :



*Le patch **PREMPT-RT** consiste en l'application d'une série de patchs au kernel ainsi qu'en la modification du fichier de configuration du noyau. Les patchs officiels pour PREEMPT-RT sont directement accessible sur kernel.org (https://www.kernel.org/pub/linux/kernel/projects/rt/). Nous constaterons que cette solution est très simple et rapide à appliquer et ne nécessite aucune couche de virtualisation supplémentaire (cf. Adeos). Le système résultant est plus léger qu'avec Xenomai/RTAI. De plus, aucune API user space spécifique n'est nécessaire (API POSIX standard), cette solution ne nécessite donc aucune modification des applicatifs déjà développés en user space.*



Se rendre sur le wiki officiel de PEEMPT-RT (https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO), récupérer la dernière version stable des patchs associée à notre release de kernel (3.8.13) et les appliquer. Se placer dans le répertoire du kernel ***~/elinux/work/kernel/kernel/*** *:*

```
cd ~/elinux/work/kernel/kernel/
wget http://www.kernel.org/pub/linux/kernel/projects/rt/3.8/stable/patch-3.8.13.13.bz2
cd kernel
bzcat ../patch-3.8.13.13.bz2 | patch -p1
```

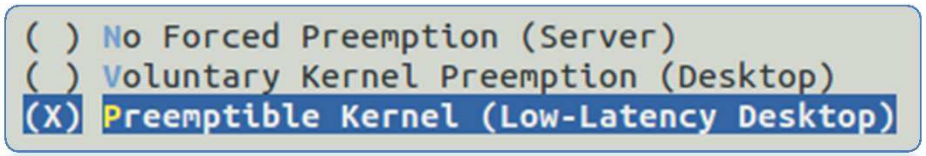
Par curiosité, extraire une archive des patchs appliqués et observer le type de modifications appliquées au code kernel :

```
--- a/arch/sparc/kernel/prom_common.c
+++ b/arch/sparc/kernel/prom_common.c
@@ -64,7 +64,7 @@ int of_set_property(struct device_node *dp,
     err = -ENODEV;

     mutex_lock(&of_set_property_mutex);
-    write_lock(&devtree_lock);
+    raw_spin_lock(&devtree_lock);
```

Nous allons maintenant modifier le fichier de configuration du kernel avant de relancer une compilation complète. Suivre le tutoriel en ligne sur le site officiel du patch PREEMPT-RT. Attention, certaines des configurations à appliquer sont dépendantes de l'architecture cible, l'exemple donné en ligne est pour une architecture x86 (architectures les mieux supportées par le patch en 2014). Activer les services suivants (interface ncurse, recherche avec /) :

- ***PREEMPT*** *: modèle de préemption du kernel*



- ***PREEMPT_RCU*** *(read Copy Update, mécanisme de type exclusion mutuelle),* ***TREE_PREEMPT_RCU***, ***RCU_BOOST***, ***RCU_BOOST_PRIO=1***, ***RCU_BOOST_DELAY=500***,

Il me reste à faire :

- *Benchmark sur jitter avec et sans charge (cyclictest et hackbench)*
- *Partie Xenomai : présentation, procédure d'installation, mise en œuvre sur BBB, benchmark, API skins, dév d'une application (CAN)*

Merci de votre attention !