J3eA - Vol. 2 - 6 (2003).
DOI : 10.1051/bib-j3ea:2003006

Programmation Temps-Réel en IUT GEII. Mini-projet sous Linux/RTAI

S. Brette [*] [¤], S. Retailleau [**] et J.-M. Siret [***] *(IUT Ville d'Avray, Paris X)*
Mis en ligne le 19 mai 2003.

Résumé

Le programme des enseignements en IUT GEII (Génie Électrique et Informatique Industrielle) inclut une partie sur la programmation multitâches/commande temps-réel. Dans cette communication, nous décrivons la mise en place de cet enseignement dans notre département sous forme classique cours/TD/TP. Cet enseignement s'appuie sur un système d'exploitation temps-réel libre et gratuit : RTAI/Linux et représente actuellement un volume d'enseignement d'une vingtaine d'heures. Le choix de ce système d'exploitation et sa mise en oeuvre pour l'enseignement ont répondu à plusieurs contraintes pratiques et pédagogiques que nous détaillons. Nous abordons les aspects matériels et logiciels concernant la mise en oeuvre pratique de cette plate forme pour l'enseignement. Enfin nous décrivons un TP de contrôle industriel utilisant le système.

Mots-clés : informatique industrielle, programmation multitâches temps-réel, Linux, RTAI.

Niveau de connaissances requis. Cours sur les systèmes temps-réel, notions utilisateur du système Unix.
Niveau des étudiants. Premier cycle (IUT GEII).

* **Stéphane Brette** [1] est professeur agrégé (PRAG) au département GEII de l'IUT de Ville d'Avray depuis 1996. Il enseigne principalement l'informatique industrielle et le traitement du signal en DUT, en licence, maîtrise (IUP).

¤ e-mail : stephane.brette@cva.u-paris10.fr (auteur correspondant)

** **Stéphane Retailleau** [1] est professeur agrégé (PRAG) au département GEII de l'IUT de Ville d'Avray depuis 1993. Il enseigne principalement l'informatique industrielle et l'électronique en DUT et en licence (IUP).

*** **Jean-Marie Siret** [1] est maître de conférences au département GEII de l'IUT de Ville d'Avray depuis 1970. Il y enseigne principalement l'informatique industrielle et l'automatique.

[1] IUT Ville d'Avray, 1 chemin Desvallieres, F-92410 Ville d'Avray, France.

1. Choix du système

Le choix d'une plateforme d'enseignement pour l'informatique industrielle/commande temps-réel (en remplacement de l'antique système DOS) nous a amené à définir un ensemble de caractéristiques, souhaitables ou nécessaires, que devait offrir ce système. En particulier :

- système d'exploitation multitâches ;
- possibilité de faire du temps-réel « dur », avec des cadences d'échantillonnage inférieures à la milliseconde pour la commande temps-réel ;
- accès possible et facile aux périphériques (port parallèle, cartes d'entrées/sorties…) ;
- fonctionnement réseau pour la gestion de comptes étudiants et la sécurisation du système. ;
- disponibilité d'une interface graphique « moderne » ;
- coût raisonnable et documentation.

L'utilisation des systèmes WinNT, WinCE et QNX a été envisagée mais ces systèmes n'ont pas été retenus car ils ne possédaient pas une ou plusieurs des caractéristiques précédemment citées. Bien que multitâches, les systèmes WinNT et WinCE ne sont pas des systèmes temps-réel.

Des cadences d'échantillonnage de l'ordre de la milliseconde ne sont pas envisageables même en l'absence de charge de la machine. Par ailleurs l'accès aux périphériques nécessite soit le développement délicat d'un pilote de périphérique, soit le fonctionnement en mode super-utilisateur, ce qui pose des problèmes de maintenance des machines.

Le système QNX [1] semblait *a priori* bien adapté. C'est un système d'exploitation multitâche temps-réel, gratuit pour les universités (mais les outils de développement sont payants). Le système fonctionne très bien en réseau. Il n'a pas été retenu pour plusieurs raisons, en particulier :

- l'absence de support des cartes graphiques de nos machines ;
- le fonctionnement temps réel, réservé au super-utilisateur (*root*) ;
- l'absence d'outils de développement simples et bon marché ;
- un manque de support et de documentation certain.

Au contraire, le système Linux est particulièrement bien adapté à l'enseignement (en plus d'être gratuit). Les sources de documentation (parfois en français [2]) sont exceptionnellement nombreuses et de qualité et une grande variété de matériel est supportée. Le fonctionnement en réseau est excellent et l'administration aisée.

Le système Linux dispose en standard de processus dit *temps-réel*. Cependant ces processus ne peuvent préempter le processeur au détriment du système. Même avec des versions du noyau Linux à faible latence [3,4] , ce mécanisme n'est intéressant que pour des cadences d'échantillonnages supérieures à la dizaine de millisecondes. L'adjonction de fonctionnalités temps-réel est possible grâce à deux variantes : RTlinux [5] et RTAI (Real time Application Interface) [6] ; nous avons retenu cette dernière. Les tâches de RTAI sont réellement temps-réel car elles préemptent le processeur au détriment du noyau Linux, ce qui n'est pas le cas pour les versions de Linux à faible latence. Cette préemption a pour effet de pouvoir utiliser des cadences d'échantillonnage relativement faibles (inférieures à 100 microsecondes voire moins), le temps de latence n'étant jamais supérieur à 10 microsecondes [3].

2. Le système RTAI/Linux

2.1. Présentation

Le système RTAI/Linux [6] est un système d'exploitation temps-réel préemptif performant. La principale particularité est qu'il fonctionne au dessus de Linux, ce dernier système étant géré comme la tâche de plus faible priorité par l'ordonnanceur temps-réel de RTAI (Fig. 1).

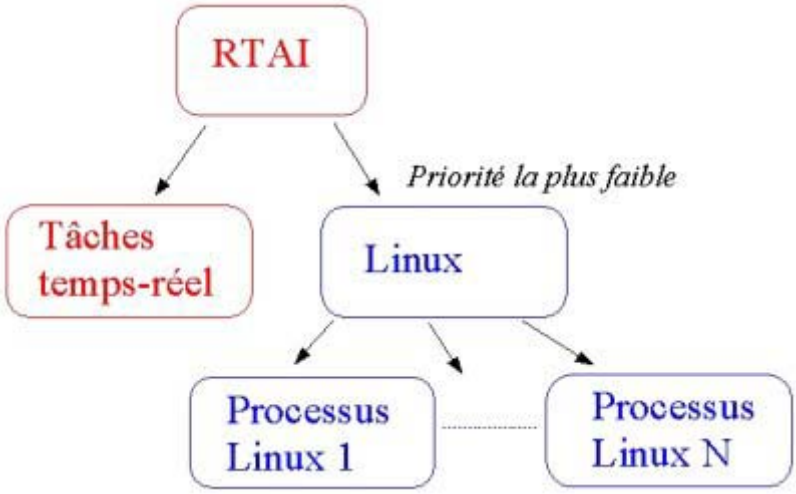


Fig. 1. Processus Linux et tâches RTAI.

Lorsque aucune tâche temps-réel n'est prête, le système Linux fonctionne normalement. Dès qu'une tâche temps-réel devient prête, l'ordonnanceur de RTAI préempte le processeur au profit de celle-ci. La gestion des interruptions matérielles est assez similaire. Les interruptions sont supervisées par RTAI et redirigées vers Linux si besoin (fonctionnement par défaut) ou traitées directement par le gestionnaire prévu par l'application temps-réel. Plusieurs moyens de communication entre tâches temps-réel et processus Linux sont disponibles : *files de messages*, *mémoire partagée*, *sémaphores*. Par ailleurs, RTAI supporte un grand nombre de mécanismes multitâches parmi lesquels on retrouve les sémaphores (exclusion mutuelle, synchronisation, ressource), messagerie, rpc, pthreads, etc.

2.2. Plateforme

La plateforme matérielle se compose de PC standards équipés de carte d'entrées/sorties analogiques et numériques et supportant une cadence d'acquisition d'environ 30 Khz, largement suffisante pour les applications envisagées. Ces machines sont connectées à un serveur de fichiers où sont localisés les comptes étudiants. Trois systèmes cohabitent : WindowsNT, Linux/RTAI et DOS et on peut « booter » sur l'un ou l'autre, au choix.

Le système Linux installé est une version Mandrake 7.0 sur laquelle on a ajouté le support temps-réel RTAI (cet ajout nécessite la recompilation d'un noyau Linux personnalisé). Cette première expérience, concluante depuis trois ans, nous incite à migrer prochainement vers des versions RedHat et RTAI plus récentes aux fonctionnalités temps-réel étendues (watchdog, etc.).

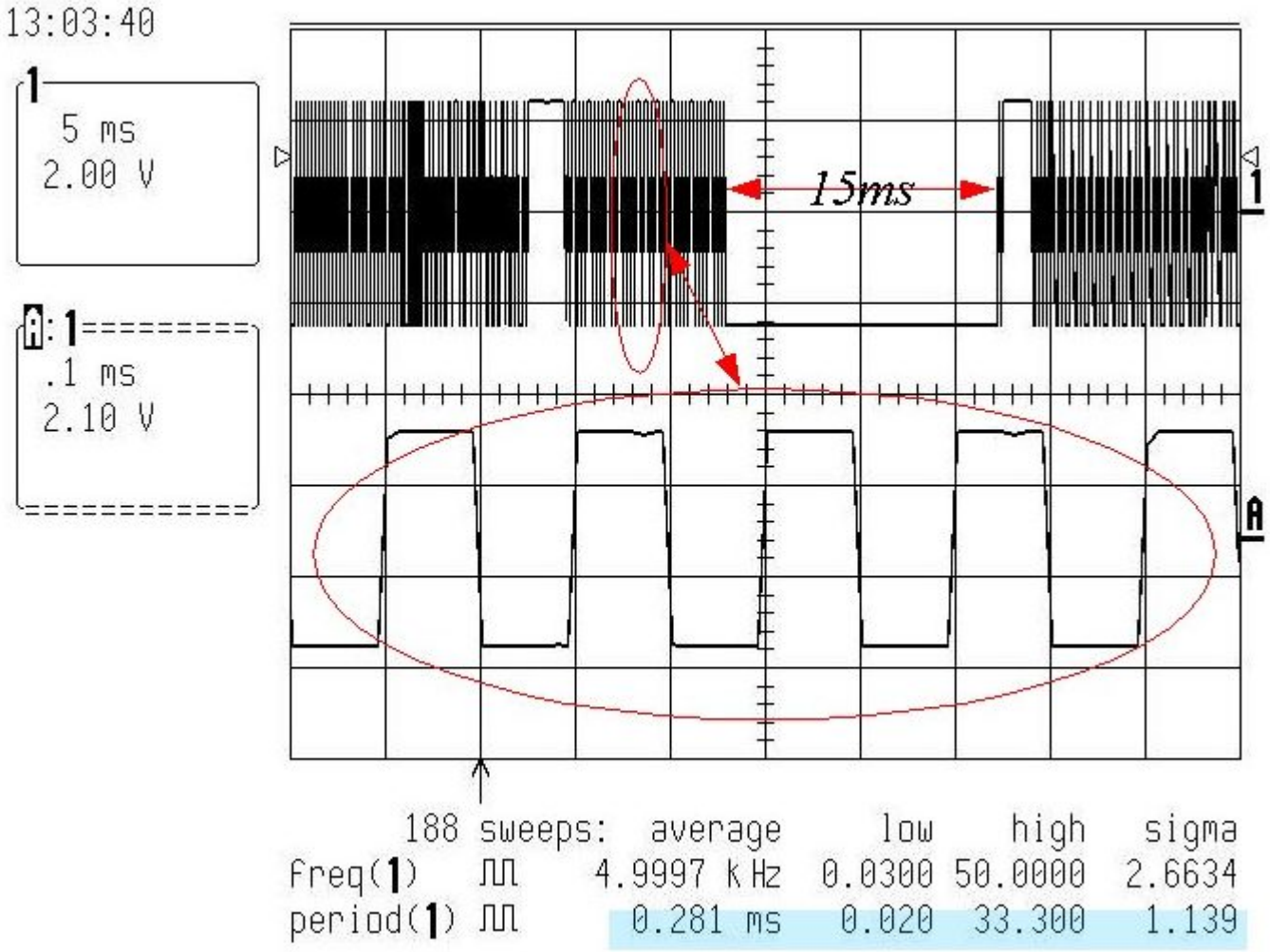
Lors des TP nous utilisons les outils standards du programmeur (gcc, make, éditeur). Par ailleurs, il n'est pas nécessaire d'être super-utilisateur (*root*) pour accéder aux périphériques et modules temps-réel. C'est un avantage important dès qu'il s'agit d'administrer le parc de machines.

3. Enseignement mis en place

L'enseignement mis en place autour de ce système se décompose en 2×2 heures de cours spécifiques à la programmation multitâches temps-réel (tâches, priorités, synchronisation, séquencement, sémaphores, messagerie). Un TD de 2 heures complète le cours, et vise à figer l'architecture logicielle du projet à réaliser en TP.

D'autres modules d'enseignement – Système Unix, programmation réseau TCP/IP, programmation orientée objet, conversions N/A et A/N – utilisent le système sans les spécificités temps-réel. Ainsi, les étudiants ayant déjà passé une quinzaine d'heures sur les postes de travail sont relativement à l'aise lors du mini-projet.

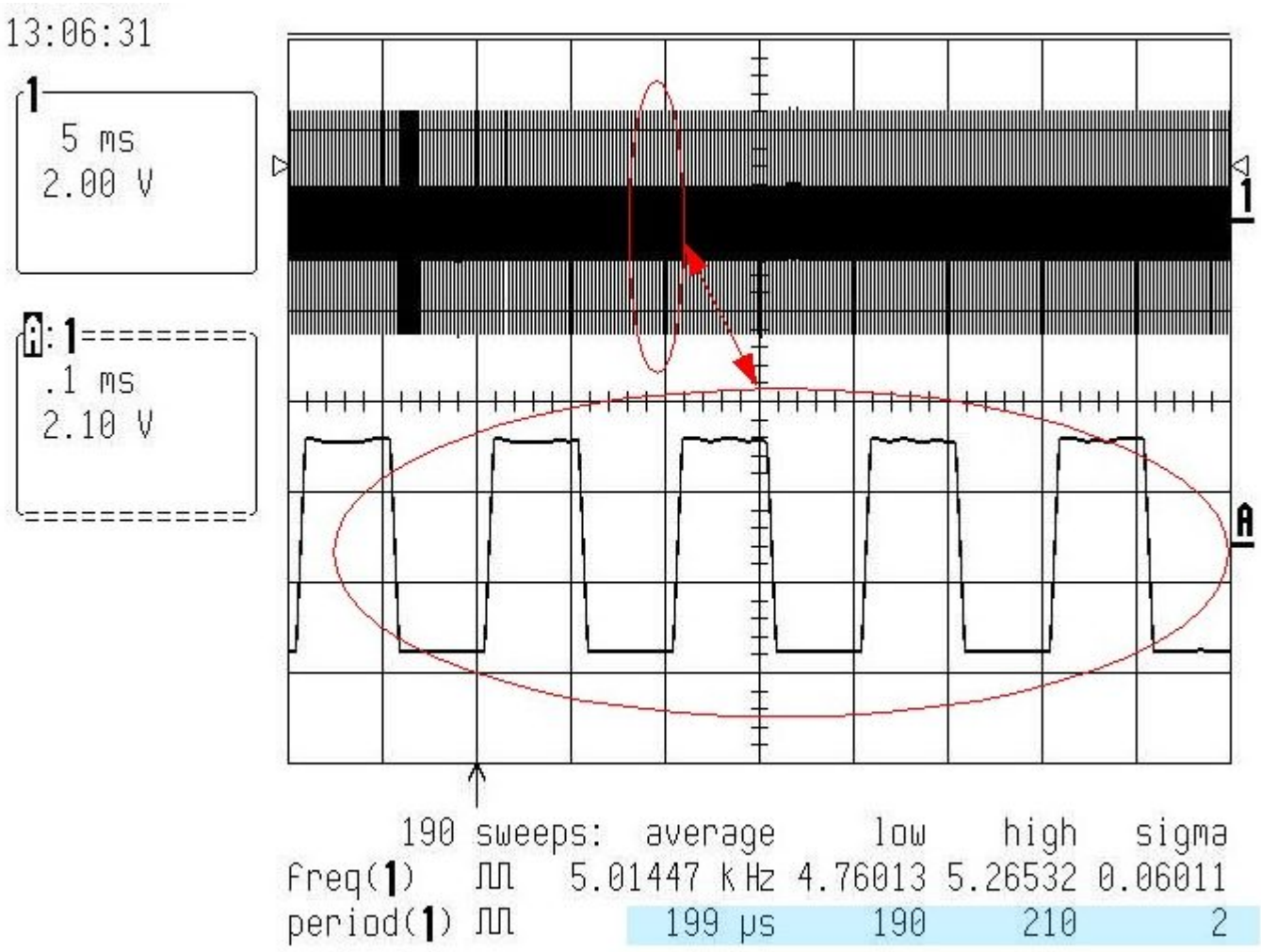
Afin d'introduire la notion de système temps réel, le TP conversion Numérique/Analogique se conclut par le relevé de l'oscillogramme (Fig. 2a), obtenu lors de la génération d'un signal carré de période 200 microsecondes resynchronisé sur un timer matériel (100 microsecondes) de la carte de conversion lorsqu'on «charge» la machine (par la commande `ls -alsR /`). Les étudiants peuvent alors comparer à celui relevé lorsqu'ils exécutent un module temps-réel fourni par nos soins (Fig. 2b) séquencé toutes les 100 microsecondes par RTAI. Même avec un système en charge, la gigue temporelle du système temps réel reste inférieure à 10 microsecondes alors que des latences de plusieures dizaines de millisecondes apparaissent sur la figure 2a.



Pseudo code figure 2a (J3eA, **Vol. 2**, art. 6)

```
init_ADA2110(10000); //10 kHz cad 0.1 ms

for(i=0;i<300000;i++){
      ecrire_CNA1(1000);
      synchro(); // resynchronisation sur front montant du timer
      ecrire_CNA1(-1000);
      synchro();
}
```



Pseudo code figure 2b (J3eA, **Vol. 2**, art. 6)

```
rt_set_periodic_mode();
period = nano2count(1e5); //1e5 = 0.1 ms
start_rt_timer(period);
rt_task_make_periodic(main_task, now, period);


for(i=0;i<300000;i++){
      ecrire_CNA1(1000);
      rt_task_wait_period();
      ecrire_CNA1(-1000);
      rt_task_wait_period();
}
```

Fig. 2a (haut) et 2b (bas). Illustration simple de la notion de temps réel.

3.1. Mini-projet proposé aux étudiants

L'objet du mini-projet (12 heures) est de réaliser un système d'asservissement échantillonné de position d'un moteur à courant continu. L'opérateur doit pouvoir à tout moment suspendre/démarrer l'asservissement et modifier les paramètres de l'asservissement (période d'échantillonnage, réglage du correcteur P.I.). Les données de l'asservissement (position, erreur, paramètres courants) sont affichées sur une console.

À l'issue du TD l'application est décomposée (Fig. 3) en trois tâches temps-réel (*Horloge, Gere, Surveille*) qui communiquent avec deux processus Linux (*Linux_gere, Linux_enregistre*) chargés des interactions avec l'opérateur (clavier, console 1 et 2).

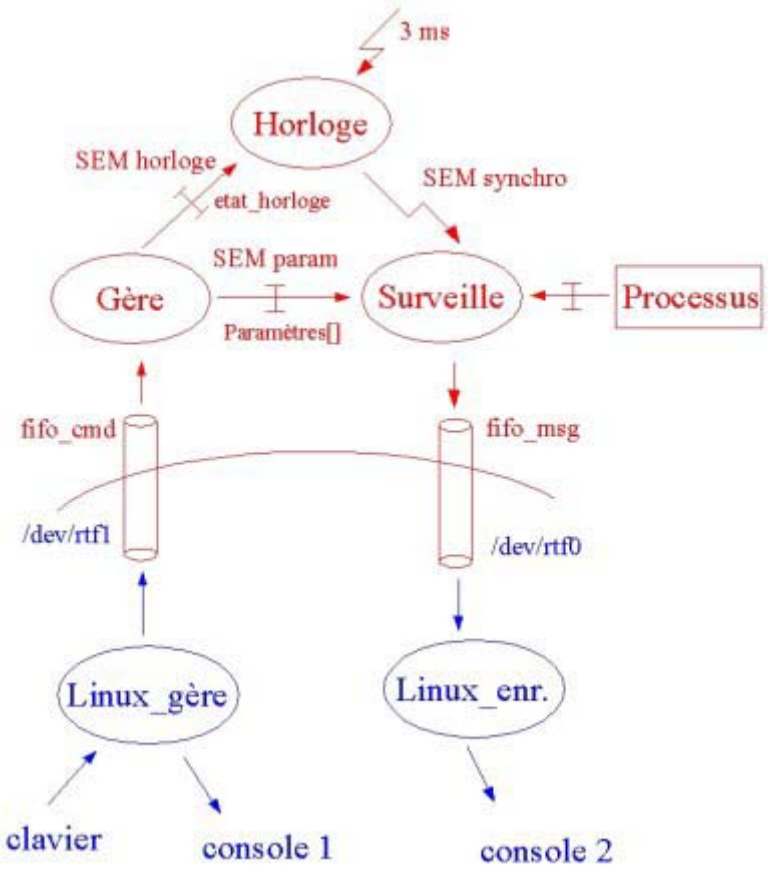


Fig. 3. Synoptique de l'application.

- La tâche *Surveille* est chargée périodiquement de mesurer la position (par le biais d'une carte d'acquisition numérique) de la comparer à la consigne (générée également par *Surveille*) pour obtenir l'erreur, de calculer et d'appliquer la commande et de déposer dans une file un message constitué de :
    - de la période d'échantillonnage ;
    - des réglages du correcteur P.I. ;
    - des mesures (erreur, position, consigne) de l'asservissement.
- Le processus Linux *Linux_Enregistre* est chargé de retirer de cette file les informations, de les mettre en forme, de les afficher sur l'écran ; ce processus affiche également l'heure de réception du message.
- L'activation périodique de la tâche *Surveille* est réalisée par la tâche *Horloge*.
- D'autre part, le processus *Linux_gere* doit, à tout moment, permettre à l'utilisateur de régler les paramètres du correcteur et la période d'échantillonnage, de démarrer et d'arrêter l'asservissement. Ce processus communique avec la tâche de supervision *Gere* par une autre file de messages.

3.2 Déroulement du projet

Le mini-projet se déroule de façon progressive et utilise un ensemble réduit de primitives RTAI (Annexe).

Dans une première phase, le module temps-réel se compose de l'unique tâche périodique *Horloge*. Ce module, entièrement fourni, permet aux étudiants d'aborder les primitives de création de tâches ainsi que la manière d'insérer des modules temps-réel sous RTAI.

Afin d'illustrer le mécanisme de synchronisation par sémaphore, les étudiants complètent le code pour ajouter une nouvelle tâche (*Surveille*) et mettre en place le sémaphore *SEMsynchro* (voir pseudo code Figure 4). C'est l'occasion d'illustrer les notions de priorités et préemption.

L'envoi des messages de défauts et la notion de files de messages (*fifos_msg*) sont introduits lors de l'ajout du processus *Linux_enregistre*.

Lorsque les étudiants complètent l'application avec le couple *Gere*, *Linux_gere*, la protection des ressources, à l'aide des sémaphores d'exclusion mutuelle (*SEMparam* et *SEMhorloge*), devient nécessaire. Cela permet la modification, sous exclusion mutuelle, des paramètres de l'application partagés par *Gere* et *Surveille*.

Au cours des manipulations, on incite les étudiants à modifier le fonctionnement de l'application en jouant sur les différents objets temps-réels présents et à interpréter les résultats obtenus. Pour ce faire, l'ajout d'indispensables `printf` judicieusement placés est complété par les informations disponibles (Tab. 1) concernant le fonctionnement de RTAI ou l'état des tâches (prête, active, endormie…).

RTAI Uniproc. Real Time Task Scheduler					
Priority	Period ns	FPU	Sig	State	Task
0	3000386	No	No	0x5	1
2	0	Yes	No	0x9	2
4	0	No	No	0x9	3

Tab. 1. État du séquenceur RTAI obtenu par la commande `cat/proc/rtai/sched`

Les tâches *Surveille* et *Gere* sont bloquées en attente de sémaphore (0x9), la tâche *Horloge* (période 3 ms) est bloquée en attente d'éveil périodique (0x5).

Ajoutons qu'il est possible, par le biais d'un moniteur d'événements LinuxTraceToolkit [7], d'enregistrer quasiment toutes les actions du système (séquencement, messagerie, préemption du processeur, etc.) avec une précision temporelle de l'ordre du temps de cycle de la machine (Fig. 4).

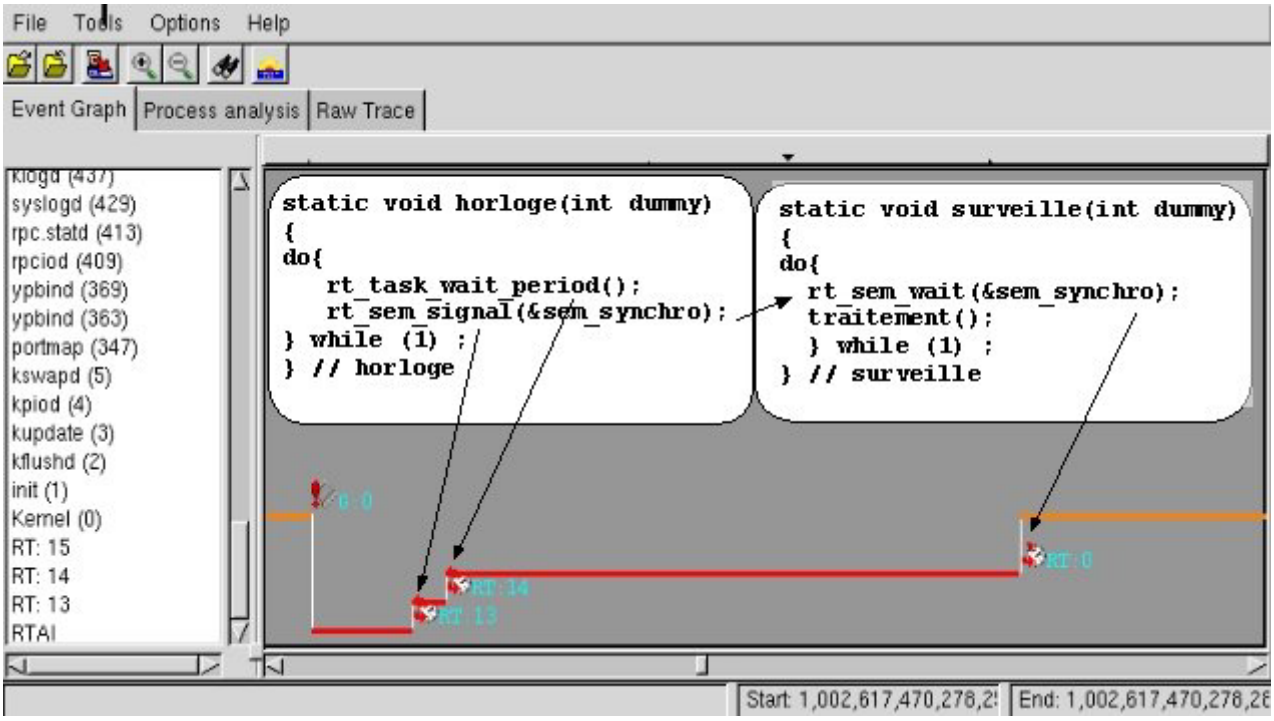


Fig. 4. Trace d'exécution sous LinuxTraceToolkit.

4. Conclusion

Nous avons mis en place un TP de programmation temps-réel développé sous le système d'exploitation temps-réel RTAI/Linux auprès d'étudiants de deuxième année d'IUT GEII. Les principaux mécanismes de la programmation multitâches temps-réel (sémaphores d'exclusion mutuelle et de synchronisation, messagerie, priorité et préemption) sont mis en oeuvre par les étudiants au cours de ce TP. Le système choisi s'est révélé particulièrement simple d'accès pour les étudiants, ce qui ne paraissait pas évident au début.

Remerciements

Nous remercions nos étudiants « cobayes » qui ont montré un réel intérêt pour cet enseignement ainsi que nos collègues pour leur investissement dans ce nouveau système.

Références bibliographiques

[1] QNX, www.qnx.com (consulté le 10 mars 2003).
[2] S. List, *Tutorial RTAI (en français)*, http://www.courseforge.org/courses/fr/rtai1/rtai.pdf (consulté le 10 mars 2003).
[3] P. Ficheux, *Linux embarqué* (Eyrolles, 2002).
[4] C. Williams, *Linux Scheduler Latency* (2002), http://www.linuxdevices.com/files/article027/rh-rtpaper.pdf (consulté le 10 mars 2003).
[5] RTLinux, http://www.fsmlabs.com (consulté le 10 mars 2003).
[6] RTAI, http://www.aero.polimi.it/~rtai/ (consulté le 10 mars 2003).
[7] K. Yaghmour, *The Linux Trace Toolkit*, www.opersys.com/LTT (consulté le 10 mars 2003).

Annexe. Primitives RTAI utilisées.

Relatives aux *tâches*	Relatives aux *semaphores*
- rt_task_init **- rt_task_delete** **- rt_task_make_periodic** **- rt_task_wait_period** **- rt_task_resume**	**- rt_sem_init** **- rt_sem_delete** **- rt_sem_signal** **- rt_sem_wait**
Relatives au *temps*	Relatives aux *files de messages*
- rt_set_periodic_mode **- start_rt_timer** **- stop_rt_timer** **- nano2count** **- rt_get_time** **- rt_sleep**	**- rtf_create** **- rtf_destroy** **- rtf_put** **- rtf_get** **- rtf_create_handler**