SETIT 2005
3rd International Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications
March 27-31, 2005 – TUNISIA

Evaluation de performances des communications asynchrones d'un système temps réel basé sur RTAI

Tarik KHOUTAIF , Fabrice PEYRARD

Laboratoire ICARE 'EA 3050'
1, Place Georges Brassens – BP 73 - 31703 Blagnac Cedex – France
khoutaif@iut-blagnac.fr
peyrard@iut-blagnac.fr

Résumé: Les systèmes temps réel constituent une partie très importante des systèmes industriels à caractère critique. Cette orientation est due à leur grande réactivité et leur réponse à la sollicitation des tâches dans des limites temporelles bien définies. Le domaine du contrôle/commande fait parti des domaines critiques où la communication entre le poste de pilotage et les robots commandés est d'une grande importance dans le sens des contraintes temporelles.
Le traitement de certaines informations doit se faire dés la réception de celles-ci. L'objectif de ce travail est d'évaluer les performances des communications asynchrones des deux modes temps réel offert par RTAI, à savoir le mode noyau et le mode utilisateur sur une plate forme basée sur RTAI comme système d'exploitation temps réel.

Mots clés: Système temps réel, RTAI, communications asynchrones, évaluation de performances.

1 Introduction

Les multiples applications robotiques nécessitent souvent des contraintes temporelles fortes. La maîtrise de cette branche scientifique nous permet d'automatiser certaines actions que l'être humain semble incapable d'accomplir avec une grande précision.

Le respect des contraintes temporelles, de la mobilité, de la fiabilité et de la disponibilité du système justifie nos activités de recherches pour l'amélioration des temps de réponse dans la boucle d'asservissement du contrôle/commande de robots mobiles à distance. Les consignes, les commandes et les réponses seront véhiculées à travers un réseau sans fil de type Bluetooth pour des transmissions à faible porté.

Pour atteindre ces objectifs, nous allons procéder par étapes. Dans la première partie de cet article, nous allons justifier le choix de RTAI (Real Time Application Interface) comme un système d'exploitation temps réel suite à un état de l'art. Ce système d'exploitation, va constituer notre plate forme temps réel. L'étape suivante sera la mise en place de RTAI sur une machine à base de processeur I386. Nous aborderons ensuite l'étude des primitives de gestion des tâches, du contrôle du matériel et plus particulièrement des ports série. Nous évaluerons les performances temporelles des tâches temps réel qui contrôlent, lisent et écrivent sur les ports séries. Nous comparerons les deux modes temps réel, `rtai_mode` et `lxrt_mode`. Ce dernier permet l'implémentation des services de RTAI dans l'espace utilisateur. Nous allons nous intéresser à la gestion des interruptions et au contrôle de l'interface RS232 pour minimiser le temps d'attente de l'information dans le buffer logiciel. Suite à cette étude (travail en cours), nous implémenterons un protocole de contrôle/commande, entre un poste de pilotage et un robot mobile, sur la base de notre système temps réel et utilisant des communications sans fil Bluetooth.

2 Les systèmes d'exploitation temps réel

2.1 Définition d'un système temps réel

Parmi les systèmes d'exploitation, nous pouvons distinguer deux familles, les systèmes d'exploitation à temps partagé (UNIX, Linux, Windows, …), et les systèmes d'exploitation temps réel. Dans un système d'exploitation à temps partagé, l'ordonnanceur partage le temps CPU entre les différentes tâches exécutables. Par contre le rôle d'un système d'exploitation temps réel, est de garantir la disponibilité des ressources matérielles pour des tâches définies, dans des limites temporelles précises. Dans un système d'exploitation temps réel, les temps d'acquisition et de traitement de l'information doivent être inférieurs aux temps

d'occurrence de cette information (P.Ficheux).

2.2 Les principaux systèmes d'exploitation temps réel non LINUX

Dans cette partie, nous présenterons un état de l'art des principaux systèmes d'exploitation temps réel (T.Khoutaif, F.Peyrard & T.Val, 2003) .
La plupart des systèmes d'exploitation temps réel sont payants, on distingue parmi eux:

- QNX qui est un micro-noyau temps réel, développé par la société canadienne QNX Software. C'est un logiciel commercial conforme à la norme POSIX, qui se rapproche du monde de « l'open-source » (QNX).
- Windows CE et Embedded Windows NT, sont conçus par Microsoft. Leur utilisation est actuellement limitée à l'équipement de nombreux PDA ou assistants personnels (Microsoft).
- Jaluna est un système d'exploitation temps réel français, construit autour du noyau libre C5 (ChorusOS 5) (Jaluna).
- eCOS (Embedded Cygnus operating System), initialement développé par la société Cygnus, et rattaché aujourd'hui à la société Red Hat Software. Ce système est bien adapté aux applications embarquées (eCOS).
- VxWorks est un exécutif temps réel qui nécessite peu d'espace mémoire. Il est développé par la société Wind River, pour des besoins de temps réel dans les systèmes embarqués (VxWorks).
- RTEMS (Time Executive for Multiprocessor Systems), est un système de compilation croisée à la norme Ada95 dédié aux systèmes embarqués temps réel (RTEMS).

Le fait que ces solutions temps réel soient payantes et dédiées à des applications spécifiques a permis l'apparition de nouveaux systèmes temps réel à base de noyau Linux.

2.3 Le temps réel dans Linux

Les développeurs des projets « open-source », ont adopté Linux pour rendre le comportement de son noyau temps réel. Pour cela, ils ont proposé deux solutions (P.Kadionik, 2002):

1. Les patchs préemptifs, qui permettent l'amélioration du comportement du noyau Linux en réduisant ses temps de latence. Cette technique permet la gestion du temps réel « mou », c'est-à-dire des applications ne nécessitant pas de fortes contraintes temporelles (période d'échantillonnage supérieur à la seconde).
2. Le noyau temps réel supplémentaire. Il s'agit d'ajouter au noyau Linux, un ordonnanceur temps réel à priorités fixes. Ce noyau supplémentaire traite directement les tâches temps réel et délègue les autres tâches au noyau Linux, ce dernier est considéré comme une tâche à faible priorité. Cette architecture permet de générer un système d'exploitation temps réel qui répond à des besoins du temps réel «dur» (A.Choquet, 2003).

Cette technologie est utilisée par RTLinux et RTAI.

2.3.1 RTLINUX

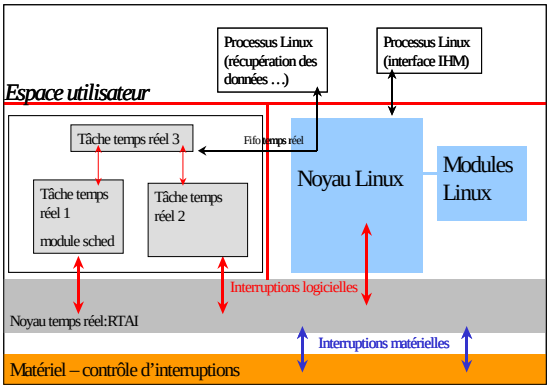
RTLinux est la première expérience à utiliser le principe du noyau temps réel supplémentaire. Il est devenu un support commercial géré par la société FSMLabs (FSMLABS).

2.3.2 RTAI

Le projet RTAI a pour origine le département d'ingénierie aérospatiale de l'Ecole Polytechnique de Milan (DIAPM) en s'inspirant de RTLinux et en intégrant quelques améliorations et corrections concernant les modes temps réel et la gestion des nombres flottants. RTAI est un logiciel temps réel « open-source », disponible sans redevance. Sa nouvelle extension LXRT (LinuX Real Time) permet de développer des tâches temps réel dans l'espace utilisateur avec une bonne granularité temporelle.

3. Structure et architecture de RTAI

Son architecture est similaire à celle de RTLinux où Linux est la tâche temps réel à faible priorité. La communication entre les différentes tâches est assurée par différents mécanismes : les FIFOs, les sémaphores les mémoires partagées, les mailbox, …. Le code de masquage et démasquage des interruptions a été réécrit (DIAPM, 2000). Les interruptions matérielles sont captées par le noyau temps réel et ne sont transmises au noyau Linux que si elles ne correspondent pas aux tâches temps réel. La figure 1 décrit cette architecture.



I. *Architecture du système RTAI*

RTAI est bâti sous la théorie des modules (S.List, 2000). rtai.o est le noyau temps réel qui gère les interruptions matérielles, rtai_sched.o qui gère les tâches, l'horloge, les sémaphores, les boîtes aux lettres et les FIFOs nécessaires à la communication entre tâches temps réel et processus Linux. Ces deux modules sont indispensables à l'exécution d'une tâche temps réel. Le module rtai_fifo.o gère des buffers uni-directionels (/dev/rtfx) de lecture/écriture pour la synchronisation et la communication entre tâches.

L'exécution de tâches temps réel dans l'espace utilisateur nécessite l'insertion du module rtai_lxrt.o qui permet l'exécution des API (Application Programming Interface) éventuellement utilisée par des tâches temps réel et des processus Linux.

Lors de l'exécution d'une tâche temps réel dans l'espace utilisateur, LXRT augmente les performances temps réel du processus en exigeant à l'ordonnanceur de changer sa politique d'ordonnancemant par la politique SCHED_FIFO.

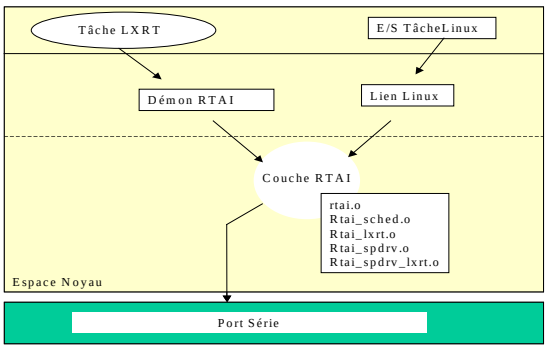
4. Le contrôle des ports série sous LXRT

Comme nous l'avons présenté, l'architecture RTAI permet deux espaces de développement, l'espace noyau et l'espace utilisateur.

Dans ce travail, nous allons étudier le contrôle des ports série dans un environnement LXRT dans un premier temps. En plus de la facilité de développement dans l'espace utilisateur, les tâches LXRT s'exécutent avec des mécanismes de protection mémoire hérités de Linux. LXRT offre aussi aux tâches une grande priorité et une meilleure granularité d'ordonnancement (V.Fias, 2002). L'exécution d'une tâche temps réel dans l'espace noyau lui offre l'accès à un espace mémoire non protégé ce qui peut rendre le système instable en cas d'erreur de programmation.

Dans cette première partie, nous avons utilisé les librairies RTAI et LXRT pour la gestion des tâches temps réel, et le contrôle des ports séries.

Dans la distribution RTAI, il existe deux formalismes pour le contrôle du port série, la technologie `rt_com` (J.Kupper, 2000) et `spdrv` (serial port drivers) (P.Montegazza, 1999). La première est la version initiale du contrôle du port série, définit par les concepteurs de RTLinux, et réutilisée par RTAI. L'environnement `spdrv` quand à lui est une version améliorée de `rt_com`, développée par les concepteurs de RTAI. Il permet en plus de l'initialisation des ports série, l'écriture et la lecture, la gestion des buffers d'émission et de réception ainsi que les interruptions. RTAI offre aux développeurs temps réel (à travers le module `rtai_lxrt_spdrv.o`) la possibilité d'utiliser `spdrv` dans l'espace utilisateur.



II. *L'accès au port série dans LXRT*

Nous présentons dans la figure 2 la situation de LXRT par rapport au noyau RTAI ainsi que l'environnement d'exécution de tâches de lecture et d'écriture sur les ports série. On y décrit également les différents niveaux logiciels séparant le matériel des tâches LXRT (FDNA, 2003).

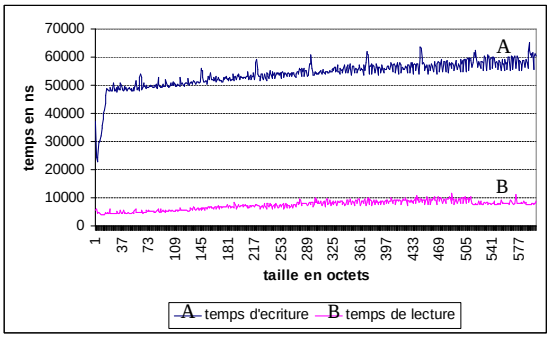
5. LXRT et les communications en mode asynchrone

5.1 Implantation

Après avoir défini l'environnement de travail, et l'avoir adapté à nos besoins, nous allons passer à une deuxième partie dans laquelle nous définissons une tâche temps réel périodique d'une grande priorité qui s'exécute d'une façon séquentielle. Cette tâche compose le message à envoyer à chaque période à partir d'une chaîne de caractères de 600 octets. A la fin de sa période elle effectue la lecture ***d***u COM2 pour extraire les données envoyées sur le COM1, acheminées par un câble croisé. Le câble croisé résout le problème de la synchronisation de d'horloge entre l'émetteur et le récepteur. En effet, nous utilisons sur notre plate-forme expérimentale une machine unique avec deux ports série, afin d'effectuer les mesures temporelles en s'affranchissant des contraintes de synchronisation multi-machines.

5.2 Performances des communications entre tâches

A partir de l'architecture décrite précédemment, nous avons pu déterminer les temps d'exécution des primitives d'écriture et lecture sur le port série dans l'environnement LXRT sur notre machine expérimentale avec une fréquence du processeur de 333Mhz. La figure 4, montre les temps d'écriture et de lecture mesurés en fonction de la taille du message transmis en octets. Le temps d'émission varie entre 30µs et 50µs en fonction de la taille des paquets [1 à 600 octets]. Le temps de réception varie sensiblement autour de 6µs quelque soit la taille des paquets.



III. *Temps d'exécution des primitives temps réel*

La primitive d'émission nécessite un temps d'exécution plus long que celle de réception, car l'écriture des octets dans le buffer logiciel temps réel s'effectue séquentiellement (octet par octet). Alors que

la réception réalise une lecture par bloc (n octets) dans le buffer de réception.

Nous avons pu quantifier le temps d'initialisation d'une tâche temps réel, qui varie entre 28µs et 30 µs. Le temps d'ouverture et d'initialisation du port série a été mesuré, et est de l'ordre de 35µs. Dans un système temps réel idéal, l'exécution des tâches temps réel n'est interrompu que par des interruptions matérielles, par contre dans notre cas (tâche LXRT), l'intervention de tâches plus prioritaires (RTAI) ainsi que les interruptions matérielles ou celles de l'horloge peuvent bien causer des retards, expliqués par les pics visibles sur la figure 4. Le module `rtai_spdrv.o` initialise (par défaut) un buffer de taille 512 octets (taille modifiable), en mode FIFO. Lors de l'utilisation de messages d'une taille supérieure à 512 octets (jusqu'à 600 dans notre cas) le système tronque à 512 octets.

5.3 Conclusion

Nous avons évaluer les communications asynchrones sous LXRT à travers le port série sur une même machine pour s'affranchir du problème de la synchronisation d'horloge. Ces travaux nous ont permis de déduire les plages temporelles pour l'écriture et la lecture sur le port série. LXRT utilise un mécanisme de boîtes aux lettres pour la communication de messages, ce qui ne permet pas de garantir le temps de la disponibilité de l'information dans le buffer. En effet, le récepteur ne dispose que d'un mécanisme périodique de consultation de la boîte aux lettres. Ce qui a pour effet, une consultation en avance ou en retard du message.

L'utilisation des interruptions va permettre de résoudre ce problème. C'est une solution de bas niveau pour la détection de l'arrivée des données sur le port série. C'est ce que nous allons présenter dans cette seconde partie.

6. La gestion des interruptions de l'UART sous RTAI

Dans cette partie nous avons utilisé la couche RTAI qui fournie des services temps réel et qui contrôle le matériel grâce à la couche RTHAL (Real Time Hardware Abstract Layer) (P.Cloutier & P.Montegazza, 2000). RTAI fournie des fonctions de gestion du port série avec `spdrv`. Il permet aussi la programmation des routines d'interruption, que nous allons utiliser pour la gestion des ports série par interruptions.

6.1 RTAI et SPDRV pour la gestion des interruptions

L'ouverture des ports série se fait à l'aide de la fonction `rt_spopen()`. Cette fonction a comme paramètre le nom du port à initialiser, la vitesse de transmission, le nombre de bits représentant un caractère, le nombre de bits de stop, la parité du protocole, le mode d'ouverture et la taille du FIFO réservé à ce port. RTAI définie trois modes d'ouverture du port série. L'UART (8250) n'émet qu'une interruption à la réception d'un caractère que s'il est ouvert en mode RT_SP_NO_HAND_SHAKE. Ce mode permet de configurer le port de façon à générer une interruption à chaque réception d'un caractère ce qui revient à mettre le bit b0 de l'adresse 0x3F8+1 à un niveau égale à 1 (La bible, 1999).

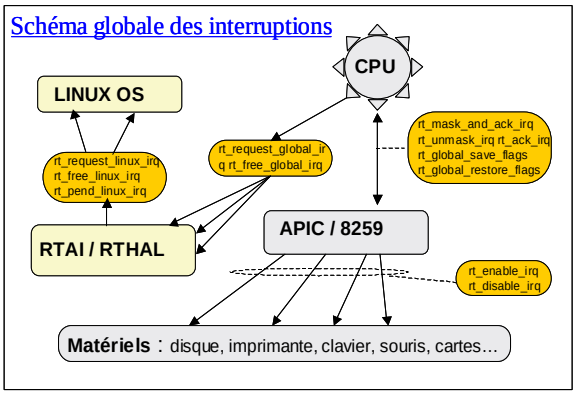
6.2 RTAI et la gestion des interruptions matérielles

Une interruption matérielle est un signal permettant à un dispositif externe au processeur d'interrompre le travail courant du processeur afin de réaliser un traitement particulier appelé routine d'interruption.

Un processeur de la famille 80386 admet 15 niveaux d'interruptions matérielles de IRQ0 à IRQ15 gérées par un circuit spécialisé appelé contrôleur d'interruptions (H.Bruyninckx & K.U.Leuven, 2002). Le contrôleur d'interruptions (8259) mémorise les requêtes d'interruptions émises par les périphériques et les délivre par priorité, sachant que l'IRQ0 est la plus prioritaire. Le contrôleur d'interruption avertie le processeur de l'arrivée d'une interruption. Ce dernier utilise le numéro d'interruption pour indexer la table des vecteurs d'interruptions contenant l'adresse des routines. Il lance alors l'exécution du gestionnaire d'interruptions. Une fois l'exécution est achevée, le contrôleur d'interruption restaure le contexte sauvegardé au moment de la prise en compte de l'interruption. Les primitives associées sous RTAI sont `rt_global_save_flags` pour la sauvegarde du contexte et `rt_global_restore_flags` pour la restauration. Une interruption plus prioritaire peut interrompre l'exécution d'un code critique. Pour cela, les interruptions sous RTAI peuvent être masquées (ou interdites) grâce aux instructions `rt_global_cli` ou `rt_global_save_flags_and_cli`. Pour autoriser ces interruptions, RTAI offre la fonction `rt_global_sti`.

L'horloge temps réel, par exemple, est l'un des périphériques qui génèrent des interruptions.

Nous décrivons dans la figure 4 les différentes entités qui participent à la gestion des interruptions dans le système RTAI. Nous y présentons également les primitives temps réel qui rentrent dans la gestion des interruptions ainsi que leurs niveaux d'interventions sur la couche logicielle.



IV. *RTAI et la gestion des interruptions*

7. Acquisition de données pour le contrôle/commande

7.1 Implantation de nouvelles routines

Les API de LXRT étant incomplètes et ne supportant pas certains services (FIFOs, mémoire partagée, et la gestion des interruptions), nous avons donc choisi de travailler dans l'espace noyau. [4].

Le développement d'une application dans l'espace noyau de RTAI fait appel à la programmation des modules. Une fois que celui-ci est chargé dans le noyau, il devient effectivement une partie intégrale du noyau. Le chargement et le déchargement se fait de deux manières, manuelle par `insmod` et `rmmod`, ou automatique grâce aux primitives `init_module()` et `cleanup_module()`. Cette dernière configuration est désactivée préalablement à l'installation pour éviter la surcharge du système.

Nous avons développé trois modules : `module_init_ports`, `module_detecte_irq` et `module_envoie`, où nous présentons ici que les caractéristiques spécifiques liées aux interruptions.

Le module `module_init_port` permet d'initialiser les ports série afin que l'UART puisse émettre une interruption à la réception d'un caractère. Le code de la primitive de chargement est détaillé ci-dessous.

```
int init_module()
{     if(rt_spopen()<0 ||
      rt_spopen()<0)
      {rt_printk( KERN_INFO
      "problems with opening SP" );}
      rt_printk(KERN_INFO "opning
      with succes");
}
```

Le module `module_detecte_irq` une fois chargé dans le noyau, permet la détection des IRQ3 et l'exécution du handler (fonction) associé. Celui-ci étant chargé de récupérer les données reçues sur le COM2. Le code ci-dessous présente l'utilisation des primitives pour référencer l'IRQ3.

```
int init_module(void)
{     rt_mount_rtai();
      rt_enable_irq(IRQ3);
      rt_request_global_irq(IRQ3,
      (void *)irq_handler);
      rt_startup_irq(IRQ3);
      rt_request_linux_irq(IRQ3,
      irq_linux_handler,
      "LNX_HNDLR",
      irq_linux_handler);
}
```

Le module `module_envoie` exécute une tâche temps réel qui envoie un message de taille définie sur le port COM1. Le code ci-dessous présente l'initialisation de la tâche ainsi que son handler.

```
static void port_serie_handler(int
t)
{     char message_env[]="bonjour,
      ici c'est ICARE !";
      rt_spclear_tx(0);
      rt_spclear_rx(1);
      rt_spwrite(0, message_env,
      sizeof(message_env));
}

int init_module(void)
{     start_rt_timer();
  rt_task_init(                    ,
port_serie_handler, , , , , );

  }
```

Une fois que le message est reçu sur le COM2, le module `module_detecte_irq` détecte uniquement la première interruption IRQ3 lors de la réception du premier caractère. `Spdrv` masque alors les interruptions suivantes. Ceci ne permettant pas de maîtriser la réception de tous les octets d'un message. Ces résultats insatisfaisants nous ont amené à voir de prés les modules `spdrv`, et de s'orienter vers une autre solution.

7.2 Conclusion de cette architecture

La couche `rtai_spdrv` capte les IRQs, les masque et exécute les routines d'interruptions définies dans le drivers `spdrv`. Ces routines redirigent les caractères reçus vers le buffer de réception propre à `spdrv.` Le module `module_detecte_irq` sensé gérer les IRQ3 entre en conflit avec le module rtai_`spdrv.o` déjà chargé dans le noyau. Pour atteindre notre objectif, nous avons deux solutions envisageables. Soit l'utilisation des fonctions spécifiques liées aux interruptions (mécanisme de `call_back`), soit développer nos propres drivers du port série qui répondraient parfaitement à nos besoins. L'objectif de nos travaux n'étant pas la réécriture des modules `spdrv`, nous avons réalisé une étude approfondie des fonctions de `call_back` qui satisfont nos exigences.

7.3 Exploitation des routines `call_back` de SPDRV

Tout en restant dans l'espace noyau, nous avons utilisé les modules `module_init_port.o`, et `module_envoie.o` présentés précédemment et un troisième module dont nous allons présenter le fonctionnement.

Ce module appelé `module_call.o` est composé des fonctions suivantes : `init_module()`, `clear_module()` et `lire_port()`. Dans `init_module()`, nous avons appelé la fonction `rt_spset_callback_fun()` qui permet de spécifier le nombre de caractères à lire et/ou à écrire sur un port série donné par un appel de handler spécifique.

Le code ci-dessous illustre l'utilisation de la fonction de `call_back` et son handler associé. Cette fonction sera donc déclanchée à chaque réception de `i` octets dans le buffer.

```
int i=1

static void lire_port()
{
  rt_spread(1, message_recu,
sizeof(message_recu));
rt_spset_callback_fun(COM2, (void
*)lire_port, i++, 1);

}

int init_module(void)
{
  rt_spset_callback_fun(COM2, (void
*)lire_port, i, 1);
 return 0;
}
```

La fonction `rt_spset_callback_fun()` exploite le mécanisme des interruptions. Une fois qu'un caractère arrive, l'interruption associée au port série utilisé est émise par l'UART. S'il n'a pas de détection d'erreurs à la réception, un compteur du nombre de caractères reçus est incrémenté. Cette technique va nous permettre la réception de l'information dés son arrivée sur le port série.

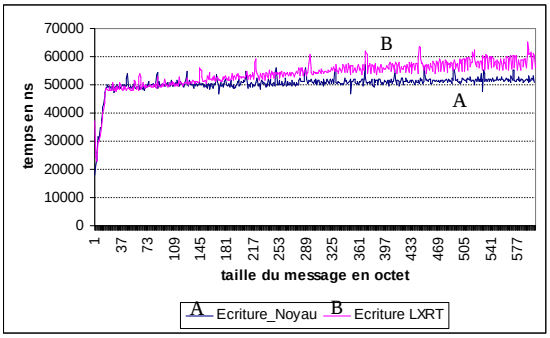
8. Mesures et résultats temporels

8.1 Comparaison entre RTAI (mode noyau) et LXRT

Les caractéristiques de nos évaluations de performances sont basées sur une taille de message variant entre 1 et 600 octets. La capture du temps est effectué en utilisant la fonction `rt_get_time_ns()` qui renvoie le temps (en ns) écoulé depuis l'initialisation de l'horloge temps réel par la fonction `rt_start_timer()`.

Nous avons évalué les temps d'exécution des primitives de lecture et d'écriture des ports série dans le mode noyau et les comparer avec les résultats obtenus en mode LXRT (Figure 3). La figure 5 représente la durée de l'écriture sous les deux modes noyau et LXRT.

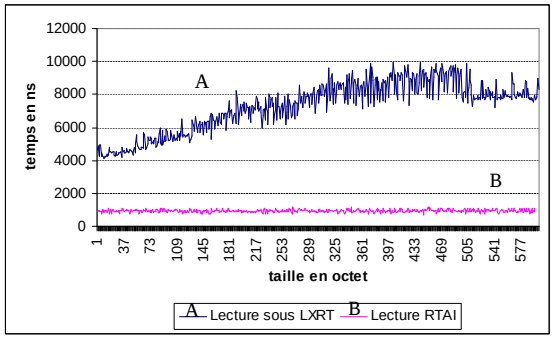
Nous pouvons constater une quasi stabilité du temps d'exécution (~50 µs) de la primitive d'écriture en mode noyau. Nous constatons aussi que ce mode noyau apporte une sensible amélioration de ce temps d'exécution.



V. *Temps d'exécution de la primitive rt_spwrite*

La figure 6 illustre une comparaison des temps de lecture dans l'espace noyau et dans l'espace LXRT.

Nous pouvons constater un temps très faible (~1µs) et une stabilité temporelle parfaite de l'exécution de la primitive de lecture dans le mode noyau. Ceci est très encourageant par rapport au mode LXRT borné par (~10µs) et relativement instable.



VI. *Temps d'exécution de la primitive rt_spread*

Suite à ces évaluations de performances, le mode noyau s'avère stable temporellement, au vu de son interaction directe avec le matériel. Cependant, ce mode reste très sensible aux éventuelles imperfections ou erreurs de programmation.

Nous allons présenter dans la dernière partie de ces travaux, les temps de transmissions sur notre plate-forme expérimentale temps réel.

8.2 Durée de transmission dans l'espace noyau

Afin de vérifier la stabilité du système global de communication, nous avons évaluer les performances de transmissions des messages dans le mode noyau uniquement. Pour ce faire, nous définissons, le temps

de transmission Ttr d'une information par la relation suivante :

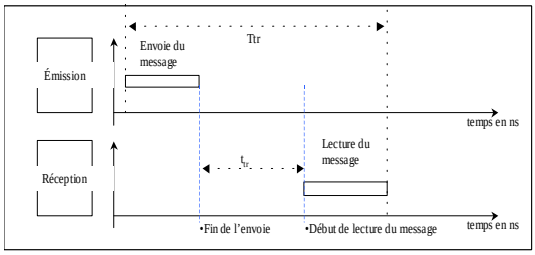
$$Ttr = t_e + t_{tr} + t_l$$

- t_e est le temps d'écriture du message sur le port.

- t_{tr} est la durée entre la fin de l'écriture sur le port série et le début de la lecture.

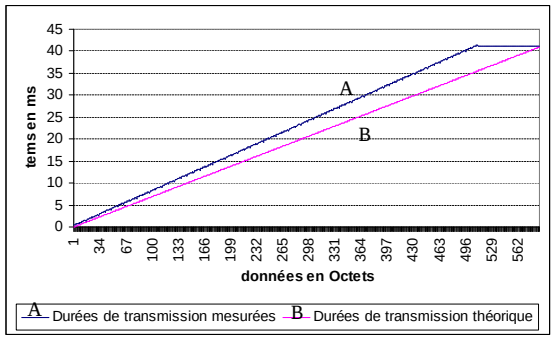
- t_l est le temps de la lecture du port série et extraction de l'information.

Le temps t_e et t_l sont déjà définis, il nous reste à déterminer le t_{tr} suivant le schéma décrit par la figure 7.



VII. *Temps de transmission à mesurer*

Sachant qu'à l'initialisation des deux ports série nous avons choisit un débit de transmission de 115200 Bauds qui est le plus grand débit offert par `spdrv`, nous avons mesuré le temps t_{tr} en fonction de la taille du message envoyé. Ce temps est illustré par la figure 9.



VIII.*Temps de transmission sur le port série*

La figure 9 montre une comparaison entre le temps de transmission théorique et celui mesuré avec un débit de 115200 bits/s et des messages de taille variant entre 1 et 600 octets. La stabilisation de la courbe des durées de transmission à partir de 512 octets est due à la taille des buffers d'émission et de réception initialisés par `spdrv`.

La courbe des valeurs mesurées est effectivement sensiblement supérieure à celle des valeurs théoriques. Cependant, ceci s'explique par le fait que les valeurs mesurées incluent les bits de gestion et de contrôle de la transmission sur le port série. Par conséquent, l'utilisation de l'ensemble des primitives dans le mode noyau montre une stabilité suffisante pour une utilisation future dans un environnement sans fil de type Bluetooth.

Conclusion

Ce travail de recherche nous a permit d'évaluer les performances des communications asynchrones sur une plate-forme temps réel.

Nous avons pu déterminer les temps d'écriture et de lecture sur le port série en mode LXRT ainsi que dans l'espace du noyau temps réel en fonction de la taille des messages à transmettre. La maîtrise du mécanisme de la gestion des interruptions sur cette plate forme temps réel de type RTAI, nous a permis de mieux gérer le temps, et de lire les messages dés leur arrivée sur le buffer du port série. Cela nous a permis aussi de vérifier la durée de transmission en fonction de la taille des message à transmettre.

Ces résultats nous ont permis de valider nos expérimentations, en s'assurant du bon fonctionnement du système temps réel RTAI choisi pour des communications filaires asynchrones. Ceci nous permet donc, de poursuivre nos travaux vers des communications sans fil Bluetooth, et de spécifier un protocole temps réel pour le contrôle/commande de robots mobiles à distances.

Références

(A.Choquet, 2003) A.Choquet, Panorama de l'ordonancement temps réel monoprocesseur, Ecole d'été temps réel 2003 Toulouse.

(DIAPM)Département d'ingénierie aérospatiale de l'école polytechnique de Milan, www.rtai.org.

(FDNA, 2003) French data network, association de développement des réseaux INTERNET et USENET http://www.fdn.fr/~brouchou/rtai/rtai-doc-prj/ (on line manual)

(J.Kupper, 2000) J. Kupper, the serial port driver of real time Linux. Institut fur Phisicalysche Chimic, Frankfurt 2000.

(La bible, 1999) La bible PC, édition Micro application.

(QNX) Le site Internet de QNX, www.qnx.com

(Microsoft) Le site Internet de Microsoft, www.microsoft.com

(Jaluna) Le site Internet de Jaluna, www.jaluna.com

(eCOS) Le site Internet d'eCOS http ://sources.redhat.com/ecos/

(VxWorks) le site Internet de VwWorks, www.windriver.com

(RTEMS) le site Internet de RTEMS, www.rtems.com

(P.Ficheux) Pierre Ficheux, Linux embarqué, édition Eyrolles.

(P.Kadionik, 2002) P.Kadionik, Introduction à linux embarqué pour vos systèmes électronique, projet

Jessica Midi-Pyrénées, 2002 LAAS-CNRS TOULOUSE.

(P.Montegazza, 1999) P.Montegazza, DIAPM for Linux : why' what's and how's, workshop 1999 vienna.

(P.Cloutier & P.Montegazza, 2000) P.Cloutier, P.Montegazza, K.yaghmour, DIAPAM-RTAI position paper, workshop RTSS 2000

(H.Bruyninckx & K.U.Leuven, 2002) Real-Time and Embedded Guide, H.Bruyninckx, K.U.Leuven, Mechanical Engineering, 2002.

(DIAPM, 2000) RTAI Programming Guide1.0, DIAPM, September 2000.

(FSMLABS) RTLinux FSlabs online, www.fsmlabs.com.

(S.List, 2000) S.List, Définition et concept de RTAI, Alcôve. France 2000, www.coursedorge.org/courses/fr/rtai

(T.Khoutaif, F.Peyrard & T.Val, 2003) T.Khoutaif, F.Peyrard, T.Val, Vers l'utilisation des RdPTS pour l'évaluation de performances d'un système de communication sans fil embarqué et temps réel dédié à la robotique, CNRIUT, Tarbes 2003.

(V.Fias, 2002) Vanessa Fias Martinez, Implementation of a signal control system in a real time environment.Computer science department of Colombia university 2002.