MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

UNIVERSITE PAUL SABATIER

TOULOUSE III

M2-EEA-ISTR-RODECO-IARF

Cahier de Travaux Pratiques

SYSTEMES
TEMPS REEL

2017-2018

PREAMBULE

Comment travailler

Attention : les modalités ci-dessous sont susceptibles d'être modifiées. Renseignez vous au début du TP sur les modalités exactes.

- Se connecter (les noms de login et les mots de passe vous seront donnés en séance) et copier dans votre répertoire, à partir du répertoire : /home/partage/commun/ RTAI-M2ISTR, tous les fichiers présents.

  ATTENTION : ne jamais modifier les fichiers d'origine dans le répertoire /home/partage/commun/M2ISTR_RTAI

- les shells "go" et "stop" permettent respectivement d'insérer et de retirer de l'espace noyau un module RTAI, un module Comedi ou un module de l'application.

- Pour chaque nouveau programme, il est nécessaire d'adapter les fichiers "Makefile", "go" et "stop"

- Une carte d'entrées/sorties NI-6221 est disponible. Pour connaître les composants (subdevice) présents sur la carte on pourra exécuter la commande :

  sudo /usr/local/src/comedilib/demo/board_info

  qui affiche sur l'écran ces informations. Pour cela il faut avoir au préalable charger les modules comedi et rtai (rtai_hal.ko et rtai_lxrt.ko) nécessaires et avoir les droits root.

TP1 : INITIATION A UN OS TEMPS REEL BASE SUR LINUX

MESURES DE PERFORMANCES TEMPS REEL

Objectifs

- Initiation à la programmation de module de noyau sous Linux
- Programmation d'application temps réel
- Communication par FIFO entre processus utilisateur et tâches noyau temps réel
- Mesures et comparaison de performances temps réel sous Linux

MAtériel utilisé

- Un PC muni de l'OS Linux - RTAI

Sujet

Le but ici est de vérifier le comportement de tâches temps réel et leur performances temporelles en présence de perturbations liées à l'exécution de tâches non temps réel.

I. Mesures sous Linux

A partir du répertoire RTAI-M2ISTR copier dans votre répertoire le dossier « tp1_mesures_TR » .
Editer le programme « ***carrelinux-comedi.c*** » et le fichier « ***Makefile1*** ».

Questions :

1. Expliquer le fonctionnement du programme « ***carrelinux-comedi.c*** ». Enregistrer ce fichier sous le nom « ***moncarrelinux.c*** ».
2. Modifier le fichier Makefile1 pour compiler votre programme « ***moncarrelinux.c*** ». Enregistrer le sous le nom Makefile.
3. Compiler le programme, copier l'exécutable dans ***/tmp***, l'exécuter avec la commande :
   sudo /tmp/moncarrelinux, et vérifier le comportement en utilisant un oscilloscope.

On souhaite mesurer les performances de linux pour une exécution temps réel. Les valeurs de période de signal seront notre moyen d'évaluation. Pour cela dans le programme on enregistre toutes les valeurs des périodes du signal.

Questions :

4. Modifier le programme « moncarrelinux.c » afin d'enregistrer chaque valeur de la période dans un tableau DELTA[i]. Lorsque on arrête le programme, par CtrlC, on enregistre dans le fichier « delta.res » les valeurs contenues dans le tableau DELTA[i], avec au maximum 5000 mesures. Pour détourner l'interruption CtrlC on utilisera l'instruction « ***signal (SIGINT, IntHandler)*** » où « ***void IntHandler (int sig) »*** est la fonction qui enregistre les valeurs dans « delta.res ».
5. Le programme (script) fourni « delta.plt » sera ensuite exécuté pour créer un tracé des valeurs contenues dans le fichier.
6. Compiler le programme, l'exécuter et vérifier le comportement en utilisant un oscilloscope ; puis exécuter « delta.plt » (en tapant **gnuplot delta.plt**) et visualiser le fichier créé delta.png. Commentez le résultat.

Pour connaître l'influence de Linux sur l'exécution de cette tâche « supposée temps réel » on charge linux en lançant divers processus, en bougeant la souris rapidement, exécutant la commande « ***cat /dev/zero > file2delete*** » ou bien ***« echo 'scale=100000;pi=4*a(1);0' | bc -l & »***.

Questions

7. Mettre linux en charge en même temps que votre programme de la question 2.

Quelles sont vos observations sur l'oscilloscope ?

8. Arrêter votre programme et la commande « cat /dev/zero > file2delete » si vous l'avez utilisée. Exécuter la commande gnuplot delta.plt et visualiser le fichier créé delta.png. Commentez le résultat. Pensez à effacer le fichier « file2delete ».

II. Mesures sous RTAI

On souhaite maintenant mesurer les performances de RTAI avec la même tâche de génération d'un signal carré.

- Copier à partir du répertoire RTAI-M2SRI puis éditer les fichiers squelet.c, Makefile-LXRT, Makefile-RTAI, go et stop.

1. Utilisation de l'environnement RTAI (Kernel) :

- Modifier le programme squelet.c pour générer le signal carré en utilisant la méthode de programmation RTAI vue en cours.
- Modifier le fichier Makefile-RTAI, le renommer « Makefile » et compiler votre programme.
- Modifier les fichiers go et stop
- Lancer votre programme par la commande « ./go » et l'arrêter avec la commande ./stop

On souhaite comparer les résultats entre les tracés des périodes du signal. L'écriture sur un fichier n'est pas possible à partir de RTAI. Il est donc nécessaire de passer par un processus Linux.

- Afin de sortir du module les mesures des périodes nous utiliserons une fifo. Modifier le module RTAI pour écrire chaque mesure dans la fifo "\dev\rtf0"
- Ecrire un processus linux permettant de lire cette fifo. Sur l'interruption "Crtl C", le processus devra écrire les valeurs dans un fichier "delta.res" contenant au maximum 5000 mesures.
- Compiler et tester votre programme.
- Comparer les résultats à ceux obtenus précédemment

2. Utilisation de l'environnement LXRT (User):

- Modifier le programme squelet.c pour générer le signal carré en utilisant la méthode de programmation LXRT vue en cours.
- Modifier le fichier Makefile-LXRT, le renommer « Makefile » et compiler votre programme.
- Lancer votre programme par la commande : ./nomdevotreprogramme ainsi que la commande « cat /dev/zero > file2delete ».
- Observer le comportement sur l'oscilloscope. Commentez le résultat en comparant à la question I.3.
- Modifier votre programme afin d'enregistrer les périodes de la tâche dans un fichier comme dans la question.

TP2 : CONCEPTION D'UN LOGICIEL TEMPS REEL

GENERATEURS DE SIGNAUX

But de la manipulation

Il s'agit de réaliser un générateur de signaux à l'aide d'un calculateur muni Linux-RTAI en optimisant les fréquences maximales accessibles tout en assurant une qualité de service "acceptable".

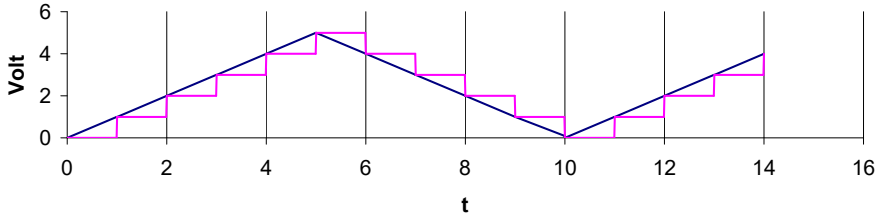
Matériel utilisé

- Un PC muni de Linux et du micro noyau RTAI
- Un boîtier d'interface PC périphérique (cf. Annexe 4 de la manipulation d'initiation) connecté à une carte d'entrée-sorties National Instrument ni-6221
- Un oscilloscope

Position du problème

L'application devra être capable de générer ***deux signaux sinusoïdaux*** bipolaires de fréquence, d'amplitude et de déphasage réglable. Les signaux générés seront délivrés par les 2 convertisseurs numérique-analogique 16 bits présents sur la carte ni-6221 et visualisés sur l'oscilloscope.

Les signaux seront approximés par un signal en escalier de ***50 échantillons par période***. La figure suivante illustre le principe de l'approximation pour un signal triangulaire :



Pour modifier les fréquences, amplitude et déphasage des signaux, on utilisera un potentiomètre délivrant une tension linéaire *Ve* variant dans l'intervalle *Vmin* = -5V et *Vmax*= +5V. La sortie de ce potentiomètre sera connectée à l'entrée d'un convertisseur analogique numérique de la carte ni-6221. On suppose également que le circuit d'entrée-sortie digital de la carte est utilisé pour sélectionner le signal à modifier et préciser quelle caractéristique est à modifier. L'affectation des bits du circuit digital est la suivante :

	b7	b6	b5	b4	b3	b2	b1	b0
Port A	X	X	X	n	a	p	f	s

Le bit *n* permet de sélectionner le signal à modifier (*n*=0 ➔ signal 0, *n*=1 ➔ signal 1). Les bits *a, p* et *f* correspondent respectivement à la sélection de la caractéristique amplitude, déphasage et fréquence. Le bit *s* correspond à la sensibilité du réglage en fréquence.

On suppose qu'au plus un seul de ces bits peut être à « un » à tout instant (dans le cas contraire on ne modifie rien). Lorsque *a*=1, une variation entre *Vmin* et *Vmax* de la tension *Ve* correspond à une variation entre

0 et 5V de l'amplitude du signal sélectionné. Lorsque *p*=1, une variation entre *Vmin* et *Vmax* de la tension *Ve* correspond à une variation entre $-\pi/2$ et $+\pi/2$ du déphasage du signal sélectionné. Enfin lors que *f*=1, une variation entre *Vmin* et *Vmax* de la tension *Ve* correspond, soit à une variation entre 1 et 100Hz de la fréquence du signal sélectionné si *s*=0, soit à une variation entre 100 et 10kHz si *s*=1.

manipulation

- Déterminer les tâches RTAI à mettre en place pour satisfaire ce fonctionnement.
- Proposer une démarche pour développer et tester progressivement l'application.
- Implémenter en respectant les différentes phases de tests et les choix de conception.
- Analyser les performances du système (fréquence max. accessible, influence de la modification d'un signal sur l'autre signal, etc.).

ANNEXE 1

Fonctions et variables utilisées dans le programme « *carrelinux-comedi.c* »

int clock_gettime(clockid_t *clk_id*, **struct timespec ***tp**);**

La fonction ***clock_gettime ()*** renvoie la valeur courante tp du temps pour l'horloge clck_id spécifiée.

int clock_nanosleep(clockid_t *clock_id*, **int** *flags*, **const struct timespec ***request**,**
struct timespec ***remain);**

Comme **nanosleep**(2), **clock_nanosleep**() permet à l'appelant de sommeiller pendant une durée indiquée avec une précision de la nanoseconde. Il diffère de **nanosleep**(2) dans le fait qu'il permet à l'appelant de choisir l'horloge avec laquelle la durée du sommeil sera mesurée et d'indiquer une valeur absolue ou relative pour la durée du sommeil.

Les valeurs de temps passées à ou retournées par cet appel sont spécifiés dans des structures *timespec* définies comme suit :

```
        struct timespec {
                time_t tv_sec;      /* secondes */
                long   tv_nsec;     /* nanosecondes [0 .. 999999999] */
};
```

L'argument *clock_id* précise l'horloge avec laquelle sera mesurée la durée du sommeil. Cet argument doit être l'une des valeurs suivantes :

CLOCK_REALTIME
Une horloge temps réel configurable à l'échelle du système.
CLOCK_MONOTONIC
Une horloge monotonique, non configurable, mesurant le temps depuis un instant du passé non spécifié qui ne change pas après le démarrage du système.
CLOCK_PROCESS_CPUTIME_ID
Une horloge configurable par processus mesurant le temps processeur consommé par tous les threads du processus.
Si *flags* vaut 0, la valeur indiquée dans *request* est interprétée comme une durée relative à la valeur actuelle de l'horloge indiquée dans *clock_id*.

Si *flags* vaut **TIMER_ABSTIME**, *request* est interprété comme un temps absolu tel qu'il est mesuré par l'horloge *clock_id*. Si *request* est inférieur ou égal à la valeur actuelle de l'horloge, **clock_nanosleep**() revient immédiatement sans suspendre le thread appelant.

clock_nanosleep() suspend l'exécution du thread appelant jusqu'à ce que soit le temps indiqué dans *request* se soit écoulé, soit un signal a été délivré provoquant l'appel d'un gestionnaire de signal ou la fin du processus.

Si l'appel a été interrompu par un gestionnaire de signal, **clock_nanosleep**() renvoie -1 et renseigne *errno* avec **EINTR**. De plus, si *remain* n'est pas NULL et si *flags* n'est pas **TIMER_ABSTIME**, il renvoie dans *remain* le temps de sommeil non consommé.

Cette valeur peut être ensuite utilisée pour rappeler **clock_nanosleep**() et achever un sommeil (relatif).

ANNEXE COMEDI

La librairie Comedi permet sous linux d'utiliser un grand nombre de cartes d'entrées/sorties avec des fonctions génériques d'acquisition d'entrée et d'écriture de sortie. Ci dessous sont données (en anglais) les fonctions principales dont vous aurez besoin pour votre manipulation.

Single digital acquisition

Many boards supported by Comedi have digital input and output channels; i.e., channels that can only produce a 0 or a 1. Some boards allow the direction (input or output) of each channel to be specified independently in software.

Comedi groups digital channels into a subdevice, which is a group of digital channels that have the same characteristics. For example, digital output lines will be grouped into a digital output subdevice, bidirectional digital lines will be grouped into a digital I/O subdevice. Thus, there can be multiple digital subdevices on a particular board.

Individual bits on a digital I/O device can be read and written using the functions

int comedi_dio_read(device,subdevice,channel,unsigned int *bit);
int comedi_dio_write(device,subdevice,channel,unsigned int bit);

The device parameter is a pointer to a successfully opened Comedi device. The subdevice and channel parameters are positive integers that indicate which subdevice and channel is used in the acquisition. The integer bit contains the value of the acquired bit.

The direction of bidirectional lines can be configured using the function

comedi_dio_config(device,subdevice,channel,unsigned int dir);

The parameter dir should be either COMEDI_INPUT or COMEDI_OUTPUT. Many digital I/O subdevices group channels into blocks for configuring direction. Changing one channel in a block changes the entire block.

Single analog acquisition

Analog Comedi channels can produce data values that are samples from continuous analog signals. These samples are integers with a significant content in the range of, typically, 8, 10, 12, or 16 bits.

The function

int comedi_data_read(comedi_t * device, unsigned int subdevice, unsigned int channel, unsigned int range, unsigned int aref, lsampl_t * data);

reads one such data value from a Comedi channel, and puts it in the user-specified data buffer.

The function

int comedi_data_write(comedi_t * device, unsigned int subdevice, unsigned int channel, unsigned int range, unsigned int aref, lsampl_t data);

works in the opposite direction. Data values returned by this function are unsigned integers less than, or equal to, the maximum sample value of the channel, which can be determined using the function

lsampl_t comedi_get_maxdata(comedi_t * device, unsigned int subdevice, unsigned int channel);

Conversion of data values to physical units can be performed by the function

double comedi_to_phys(lsampl_t data, comedi_range * range, lsampl_t maxdata);

There are two data structures in these commands that are not fully self-explanatory:

comedi_t: this data structure contains all information that a user program has to know about an open Comedi device. The programmer doesn't have to fill in this data structure manually: it gets filled in by opening the device.

lsampl_t: this "data structure" represents one single sample. On most architectures, it's nothing more than a 32 bits value. Internally, Comedi does some conversion from raw sample data to "correct" integers. This is called "data munging".

Voici par exemple un programme pour faire une acquisition sur une entrée (digital ou analogique)

```
#include <stdio.h>           /* for printf() */
#include <comedilib.h>
int subdev = 0;              /* indique quel composant de la carte sera
utilisé */
int chan = 0;           /* indique quel port ou voie sera utilisé */
int range = 0;          /* indique l'échelle qui sera utilisée /*
int aref = AREF_GROUND; /* indique la référence de masse à utiliser /*


int main(int argc,char *argv[])
{       comedi_t *ma_carte;
        lsampl_t data;

        ma_carte=comedi_open("/dev/comedi0");   (Ouverture du device
                                                comedi correspondant à la
                                                carte d'interface)-
        comedi_data_read(ma_carte,subdev,chan,range,aref, & data);
        /*lecture d'une donnée contenue dans la variable data, sur
        ma_carte, sur le composant 0, la voie 0, la première échelle 0,
        utilsant la référence AREF_GROUND, */

        printf("%d\n",data);
        return 0;
        }
```

Pour l'écriture il suffit de remplacer read par write

```
comedi_data_write(ma_carte,subdev,chan,range,aref, & data);
```