HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & embedded Digital Systems

Prof. Yann Thoma

Laboratoire de Programmation Temps Réel

semestre automne 2020 - 2021

Laboratoire 2 : Limites de Linux dans un contexte temps réel

Temps à disposition : 4 périodes

Laboratoire individuel

Objectifs

Ce laboratoire va permettre d'expérimenter l'interface des timers POSIX et les limites de Linux en tant que système temps réel.

Développement

Afin de nous familiariser avec le développement sous Linux, nous allons écrire, compiler et exécuter un petit logiciel.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

#define NB_MESURES 30

int main (int argc, char **argv)
{
    struct timeval tv;
    int i;

    for (i = 0; i < NB_MESURES; ++i) {
        gettimeofday(&tv, NULL);
        printf("%2d : %ld.%06ld\n", i, tv.tv_sec, tv.tv_usec);
    }
    return EXIT_SUCCESS;
}
```

1. Créez le fichier gettimeofday.c
2. Copiez le programme ci-dessus dans le fichier et enregistrez-le
3. Compilez avec la commande
   `gcc -o gettimeofday -Wall gettimeofday.c`
4. Exécutez avec la commande `./gettimeofday`
5. Vous pouvez également rediriger la sortie vers un fichier :
   `./gettimeofday > data.txt`
6. Vous pouvez ensuite le lire avec `less data.txt` (pressez 'q' pour quitter)

Modification

1. Copiez l'ancien fichier avec
   `cp gettimeofday.c gettimeofday2.c`
✠ 2. Modifiez le fichier gettimeofday2.c de manière à utiliser un tableau
   `struct timeval tv[NB_MESURES];`
   pour y stocker les valeurs de temps. Utilisez également deux boucles : une première pour appeler `gettimeofday()` et une deuxième pour imprimer les valeurs obtenues
✓ 3. Comparez les résultats avec les précédents
✓ 4. Expliquez pourquoi il y a des différences
✓ 5. Quelle est la granularité, ou précision, de la fonction `gettimeofday()` ?

Horloges Posix

Nous avons à présent utilisé `gettimeofday()` pour mesurer des temps et lors du précédent labo nous avons utilisé `time()`. Nous allons maintenant tester un autre mécanisme offert par POSIX.

```
int clock_gettime (clockid_t which_clock, struct timespec *tp);
int clock_getres (clockid_t which_clock, struct timespec *tp);
```

Extrait du manuel (man) :

```
DESCRIPTION
       clock_gettime  returns  the current timespec
       value of tp for the specific clock, which_clock.
       The values that clockid_t currently  supports for
       POSIX.1b  timers, as defined in
       include/linux/time.h, are:

       CLOCK_REALTIME
            Systemwide realtime clock.

       CLOCK_MONOTONIC
            Represents monotonic time. Cannot be set.

       CLOCK_PROCESS_CPUTIME_ID
            High resolution per-process timer.

       CLOCK_THREAD_CPUTIME_ID
            Thread-specific timer.

       CLOCK_REALTIME_HR
            High resolution version of CLOCK_REALTIME.

       CLOCK_MONOTONIC_HR
            High resolution version of CLOCK_MONOTONIC.

RETURN VALUE
       clock_gettime returns 0 on success; otherwise,
       it returns one of the errors listed in the
```

```
       "Errors" section.

ERRORS
       -EINVAL
            An invalid which_clock value was specified.
       -EFAULT
            An invalid pointer was specified for tp.
```

Reprenez le programme `gettimeofday.c` du premier point, et modifiez-le pour utiliser les horloges POSIX. Comparez-les en termes de résolution d'horloge.

✓✠ Commentez votre code et vos résultats.

Développement : timers

Etudiez le code suivant, tiré de

`http://www.informit.com/articles/article.aspx?p=23618&seqNum=14`

```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

void timer_handler (int signum)
{
    static int count = 0;
    printf ("timer expired %d times\n", ++count);
}

int main (void)
{
    struct sigaction sa;
    struct itimerval timer;

    /* Install timer_handler as the signal handler for SIGVTALRM. */
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = &timer_handler;
    sigaction(SIGVTALRM, &sa, NULL);

    /* Configure the timer to expire after 250 msec... */
    timer.it_value.tv_sec = 0;
    timer.it_value.tv_usec = 250000;
    /* ... and every 250 msec after that. */
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 250000;
    /*
     * Start a virtual timer. It counts down whenever this process is
     * executing.
     */
    setitimer(ITIMER_VIRTUAL, &timer, NULL);

    /* Do busy work. */
    while (1);
}
```

Attention, sous Xenomai 2.6 l'option `ITIMER_VIRTUAL` ne fonctionne pas, il faut utiliser `ITIMER_REAL` et attendre le signal `SIGALRM`.

✓ Pouvez-vous expliquer comment il fonctionne ? (pour quitter utilisez ctrl+c, ou envoyez un signal au processus avec la commande `kill`).

Modifications

Ecrivez un programme qui prend en entrée le nombre de mesures à faire et un temps en microsecondes. Programmez un timer périodique `CLOCK_REALTIME` qui affiche sur la sortie standard le temps écoulé entre deux différentes occurences. Observez le code suivant pour y prendre quelques ✓✠ idées.

```
// Configurer le timer
signal(SIGRTMIN, handler_signal);
event.sigev_notify = SIGEV_SIGNAL;
event.sigev_signo  = SIGRTMIN;
nsec = msec * 1000;  // en nanosec
spec.it_interval.tv_sec  = nsec / 1000000000;
spec.it_interval.tv_nsec = nsec % 1000000000;
spec.it_value = spec.it_interval;

// Allouer le timer
timer_create(CLOCK_REALTIME, &event, &timer);

// Programmer le timer
timer_settime(timer, 0, &spec, NULL);
```

Mesures

Maintenant que nous avons un programme implémentant un timer périodique, nous allons l'utiliser pour mesurer les capacités du système à générer des évènements périodiques.

```
$ ./timer 1000 500 > t500.dat
```

Compilez le programme fourni et utilisez-le de la manière suivante :

```
$ gcc -O2 summary1.c -o summary -lm
$ ./summary < t500.dat
```

✓ Commentez les résultats et répétez l'expérience avec un intervalle de $1ms$ et $250us$.

Le programme `octave` permet de jeter un oeil à l'histogramme des fréquences des données récupé- ✓ rées :

```
$ octave
  octave:1> help hist
  octave:2> help load
  octave:3> help exit
```

Perturbations

Répétez l'expérience avec un timer de $1ms$ en perturbant le système des manières suivantes :

1. En le lançant avec différentes valeurs de `nice`
2. En le lançant avec `./cpu_loop & ./timer 1000 1000`
3. En le lançant et en effectuant d'autres opérations avec le système en parallèle
4. En le lançant en même temps que vous recevez un `ping` depuis une autre machine (demandez de l'aide à un autre groupe)

5. *Optionnel* : les mesures dépendent de l'ordonnanceur du noyau ; répétez quelques expériences avec différentes versions du noyau, sans ou avec les patchs temps réel, et comparez les résultats.

✓✠ Commentez votre code et les résultats obtenus.

Travail à effectuer

Un petit rapport au format PDF devra être remise sur cyberlearn au plus tard avant le début de la séance suivante, de même que vos fichiers sources. Le rapport devra contenir les réponses aux questions directes (indiquées par le symbole ✓). Les connaissances acquises durant les laboratoires seront également testées lors des tests écrits ainsi que lors de l'examen final.

— Rendez le tout sur cyberlearn, dans un fichier compressé nommé `rendu.tar.gz`
  — Ce fichier doit être généré en lançant le script `ptr_rendu.sh`
  — Le script vérifie qu'un fichier `rapport.pdf` est présent à son niveau, ainsi qu'un dossier `code` également présent au même niveau