

## Laboratoire de Programmation Temps Réel

### semestre printemps 2023 - 2024

## Laboratoire 2 : Limites de Linux dans un contexte temps réel

Temps à disposition : 4 périodes

Laboratoire individuel

### Objectifs

Le premier laboratoire vous permettait de vous familiariser avec les processus et les threads sous Linux. 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 compiler et exécuter ce petit logiciel :

```
#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. gettimeofday.c vous est fourni. Compilez et déployez le sur la DE1 :  
make all TARGET=gettimeofday  
make deploy TARGET=gettimeofday
2. Connectez-vous sur la DE1 via un secure shell :  
ssh root@192.168.0.2
3. Exécutez votre programme sur la DE1 :  
./gettimeofday
4. Vous pouvez également rediriger la sortie vers un fichier :  
./gettimeofday > results.txt
5. Vous pouvez ensuite le lire avec less results.txt (passez 'q' pour quitter) ou récupérer vos fichiers localement avec secure copy scp root@192.168.0.2:/root/results.txt .

## Modification

---

- ✂ 1. 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
- ✓ 2. Comparez les résultats avec les précédents
- ✓ 3. Expliquez pourquoi il y a des différences
- ✓ 4. 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 `signal_timer.c` fourni. Cet exemple est tiré de

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

```
#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);
}
```

✓ 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 `signal_timer2.c` 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 occurrences. 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 événements périodiques.

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

Compilez le programme fourni sur votre machine de travail 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 *250us*, *1ms* et *10ms*.
- ✓ A partir des résultats précédents, créez les histogrammes des fréquences des données récupérées. Vous êtes libre d'utiliser votre environnement préféré (python, excel, octave, etc...).

## 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)
- ✓ Décrivez et commentez 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