heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

PTR

LABORATOIRE 03

Loïc Haas

13 octobre 2015

Table des matières

1 Introduction

Le but principal du laboratoire est de se familiariser avec les taches Xenomai et de comparer la précision des tache periodiques entre PTHREAD et XENOMAI lorsque le system est surchargé. Toutes les mesures effectuée dans ce laboratoire sont sur un seul cœur CPU avec l'application *dohell* exécutée simultanément.

2 Résultats

2.1 Pthread

Voici les résultat de l'intervalle entre deux exécutions de la tache periodique que l'on obtiens avec le code page 3 pour 10000 mesures avec un intervalle d'une milliseconde les valeurs sont donné en nanosecondes.

```
Total of 10000 values
  Minimum  = 1005730.000000 (position = 6750)
  Maximum  = 383418316.000000 (position = 2049)
  Sum      = 10890408840.000000
  Mean     = 1089040.884000
  Variance = 14619504619410.791016
  Std Dev  = 3823546.079154
  CoV      = 3.510930
```

2.2 Xenomai

Voici les résultat de l'intervalle entre deux exécutions de la tache periodique que l'on obtiens avec le code page 4 pour 10000 mesures avec un intervalle d'une milliseconde les valeurs sont donné en nanosecondes.

```
Total of 10000 values
  Minimum  = 992747.000000 (position = 5609)
  Maximum  = 1007146.000000 (position = 5608)
  Sum      = 9999999385.000000
  Mean     = 999999.938500
  Variance = 247435.016846
  Std Dev  = 497.428404
  CoV      = 0.000497
```

2.3 Comparaison des résultats

On vois bien que l'écart type d'une tache XENOMAI ($0.497\mu s$) est bien plus petit que celui d'une tache PTHREAD ($3.824ms$) avec un rapport de 7000 on se rend bien compte que les taches XENOMAI ne sont presques pas perturbée par les application s'exécutant dans le user space.

3 Code

3.1 Pthread

Listing 1 – pthread_timer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#include <pthread.h>

#define CLOCK_TYPE CLOCK_REALTIME

// Structure pour passer des donnee au thread
typedef struct task_data {
    struct timespec sleep_time;
    unsigned long nb_loop;
} task_data_t;

// Fonction executee par le thread
void *do_some_loop(void *cookie) {
    task_data_t *data = (task_data_t *)cookie;
    struct timespec ts;
    struct timespec last_ts;
    long long time;
    long long last_time;
    unsigned long i;

    // Effectue la tache toutes les temps donne
    clock_gettime(CLOCK_TYPE, &last_ts);
    last_time = last_ts.tv_sec * 1000000000 + last_ts.tv_nsec;
    for (i = 0; i < data->nb_loop; ++i) {
        nanosleep(&(data->sleep_time), NULL);
        clock_gettime(CLOCK_TYPE, &ts);

        time = ts.tv_sec * 1000000000 + ts.tv_nsec;

        printf("%lld\n", time - last_time);
        last_ts = ts;
        last_time = time;

    }
    return NULL;
}

int main(int argc, char *argv[]){
    unsigned long usec_sleep;
    task_data_t t_data;
    pthread_t thread;

    if (argc < 3){
        printf ("Require argument : nb_loop, sleep time in ns\n");
        return EXIT_FAILURE;
    }

    // Recupere les parametres
    usec_sleep = atol(argv[2]);
    t_data.sleep_time.tv_sec = usec_sleep / 1000000000ul;
    t_data.sleep_time.tv_nsec = usec_sleep % 1000000000ul;

    t_data.nb_loop = atol(argv[1]);

    // Cree le thread
    if (pthread_create(&thread, NULL, do_some_loop, &t_data)) {
        fprintf(stderr, "Faild to create thread");
        return EXIT_FAILURE;
    }

```

```
    // Attend la fin de l'execution
    if (pthread_join(thread, NULL)) {
        fprintf(stderr, "Can not join thread");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

3.2 Xenomai

Listing 2 – xenomai_timer.c

```
#include <rtdk.h>
#include <native/task.h>
#include <native/timer.h>
#include <time.h>
#include <stdlib.h>
#include <sys/mman.h>

// Structure pour passer des donnee a la tache Xenomai
typedef struct task_data {
    RTIME  period;
    unsigned long nb_loop;
} task_data_t;

// Fonction executee en tant que tache xenomai
void do_some_loop(void *cookie) {
    task_data_t *data = (task_data_t *)cookie;
    RTIME ts;
    RTIME last_ts;

    last_ts = rt_timer_read();
    rt_task_set_periodic(rt_task_self(), TM_NOW, data->period);
    unsigned long i = 0;

    // Effectue la tache periodique n fois
    while (0 == rt_task_wait_period(NULL)) {
        ts = rt_timer_read();

        rt_printf("%ld\n", ts - last_ts);
        last_ts = ts;
        if (++i == data->nb_loop) return;

    }
}

int main(int argc, char *argv[]){
    // Desactive le SWAP pour lapp
    mlockall(MCL_CURRENT | MCL_FUTURE);
    // Initialise rt_printf
    rt_print_auto_init(1);

    task_data_t t_data;
    int error;

    RT_TASK task;

    if (argc < 3){
        rt_printf ("Require argument : nb_loop, sleep time in ns\n");
        return EXIT_FAILURE;
    }

    // Recupere les arguments
    t_data.period = atol(argv[2]);
    t_data.nb_loop = atol(argv[1]);

    // Cree et demare la tache Xenomai
    if ((error = rt_task_spawn(&task, "First timer task", 0, 99, T_JOINABLE, do_some_loop
        , &t_data)) != 0) {
        rt_fprintf(stderr, "rror: cannot create the task (%s)\n", strerror(-error));
        return EXIT_FAILURE;
```

```
59      }
60
61      // Attend la fin de la tache
62      if (rt_task_join(&task)) {
63          rt_fprintf(stderr, "Can not join task");
64          return EXIT_FAILURE;
65      }
66      return EXIT_SUCCESS;
67  }
```