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

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Laboratoire 03 - PTR

Linux et Xenomai

Auteur:
Denis Bourqui
Professeur:
Yann Thoma

Salle:
A07

Priorité des threads et Linux

Sans dohell on arrive à une vitesse de boucle de 445'000'000 cycles par secondes. Avec le dohell par contre on est pratiquement à la moitié. (200'000'000 cycles par secondes). Ceci semble assez logique car il y a plusieurs processus qui doivent travailler sur un seul precesseur.

Mesure d'une tache périodique avec nanosleep

Une tâche à haute prio faisant une boucle qui nanosleep 100ms à chaque itèration arrive env **50 à 100 microsecondes** en retard à chaque itération sur un processeur qui est à 100%.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <inttypes.h>
#include <errno.h>
#include <sys/time.h>
#include <time.h>
#include <string.h>

#define EXECUTION_TIME 10
#define PERIOD_MS 100



int main(void){
    struct timespec lastOccTime;
    struct timespec actOccTime;
    struct timespec waittime;
    waittime.tv_sec  = 0;
    waittime.tv_nsec = PERIOD_MS * 1e6;

    time_t start_time = time(NULL);
    clock_gettime(CLOCK_REALTIME, &lastOccTime);
    //unsigned long nb_iterations = 0;
    while (time(NULL) < (start_time + EXECUTION_TIME)) {
        nanosleep(&waittime, NULL);

        /* get time */
        clock_gettime(CLOCK_REALTIME, &actOccTime);

        /* calculate time difference between last cycle */
        long long timediff_s = actOccTime.tv_sec - lastOccTime.tv_sec;
        long long timediff_ns = actOccTime.tv_nsec - lastOccTime.tv_nsec;
        unsigned long long timediff = (timediff_s * 1e9) + timediff_ns ;

        fprintf(stdout ,"cycle came after %lld nanoseconds\n", timediff);

        clock_gettime(CLOCK_REALTIME, &lastOccTime);

    }
}
```

Xenomai

Une tâche à haute prio qui se reveille périodiquement après 100ms arrive **10 microsecondes en retard ou en avance sur un processeur qui est sur 100% avec xenomai.
On voit qu'on arrive à etre plus précis en utilisant xenomai qu'un sleep normale.

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


#define TIMESLEEP 100000000 /* 0.1 sec */
#define NB_PERIODS 15

RTIME starttime;
RT_TASK mytask;

void task_func(void *arg){
    RTIME now;
    RTIME last;

    /* set priod */
    if(rt_task_set_periodic(rt_task_self(), TM_NOW,
rt_timer_ns2ticks(TIMESLEEP))){
        rt_printf("error on set period\n");
        return;
    }

    last = rt_timer_read();
    for(int i = 0; i < NB_PERIODS; ++i){
        // get time
        now = rt_timer_read();

        // calculate delta
        int delta  = now - last - TIMESLEEP;
        rt_printf("time: %lluns, delta %d: %dns\n", now, i, delta);
        last = now;

        //wait on next period
        rt_task_wait_period(NULL);
    }

}


int main(void){
    rt_printf("go\n");

    // init time 0
    starttime = rt_timer_read();

    // taskname
    const char *taskname = "_mytask";

    // memory lock
    if(mlockall(MCL_CURRENT|MCL_FUTURE)){
```

```
        rt_printf("mlockall failed\n");
        exit(1);
    }

    rt_print_auto_init(1);

    // create joinable task running with high priority
    rt_printf("create\n");
    if(rt_task_create (&mytask, taskname, 0, 90, T_JOINABLE)){
        rt_printf("error on task create");
    }

    // run task
    rt_printf("start\n");
    if(rt_task_start (&mytask, &task_func, NULL)){
        rt_printf("error on task start");
    }

    // wait on task to finish
    rt_printf("wait on join\n");
    rt_task_join(&mytask);
    rt_printf("joined\n");

}
```