Lab 2 - Threads and Real Time Tasks

Week 2 – Real Time Tasks in User Space

ECE 4220/7220
Real Time Embedded Computing
Spring 2016
University of Missouri
What is a Linux scheduler?

- An important goal of a scheduler is to allocate CPU time slices efficiently while providing a responsive user experience.

- A CPU can be considered a resource to which a scheduler can temporarily allocate a task (in quantities called slices of time).

- The scheduler makes it possible to execute multiple programs at the same time, thus sharing the CPU with users of varying needs.
Why Use RTAI for Scheduling?

- Processes are not limited by the general purpose GNU/Linux scheduler
- Processes can be executed precisely as designed
- Meet hard real time constraints
Schedulers

- RTAI has two schedulers
  - rtai_lxrt
  - rtai_sched

- rtai_lxrt is a Linux co-scheduler which supports hard real time for Linux schedulable objects (like processes, threads)
- rtai_sched schedules kernel tasks in addition to those mentioned above
Schedulers

Kernel Space

RT
T1

RT
Tn

Sched / OS / Kernel

User Space RT

FIFO

RT Scheduler (rt_sched)

rtai_fifo

rtai_lxrt

Hardware
rtai_sched

- In early days only rtai_sched was available and was suitable for the scheduling of RTAI lightweight kernel tasks.
- Every other activity within Linux which doesn't use any of the RTAI functionality (either kernel space or user space) is scheduled by the native Linux scheduler.
**rtai_lxrt**

- This scheduler is capable of providing hard real time for all Linux schedulable objects, but it’s not able to schedule the RTAI lightweight kernel tasks.

- When creating a real time task in user space with the LXRT API some kind of a hard real time shadow is created for being scheduled by the RTAI scheduler.

- The communication between the user space application and the shadow thread is the overhead incurred by LXRT but we have guaranteed execution times with slightly worse performance.

- If a user space LXRT application does a Linux System Call the application looses its hard real time capabilities until the return from Linux.
Differences Between Schedulers

- Kernel tasks scheduled with rtai_sched have a faster switching time
- However these tasks operate outside the Linux environment
- This can cause problems if you need to work with processes within Linux
- Interrupts are easier to work when using a kernel task
RTAI Types

- **RT_TASK**
  - Task structure to hold information about the task
  - Passed to many functions in the API

- **RTIME**
  - RTAI's timer is based off of 'counts'
  - Counts are different from nanoseconds and must be converted using nano2count or count2nano
Common Functions

RT_TASK* rt_task_init(unsigned long name, int priority, int stack_size, int max_msg_size)
  - This function extends the linux task structure allowing use of RTAI functions
  - For unsigned long name use the nam2num function

RTIME start_rt_timer(int period)
  - Starts the timer, period must be in counts (use nano2count())
  - Only needs to be called once

void stop_rt_timer(void)
  - Stops the timer

int rt_task_make_periodic (RT_TASK *task, RTIME start_time, RTIME period)
  - Marks task for periodic scheduling
  - Start time is absolute: usually done by calling rt_get_time()+offset
  - Period is a multiple of the period returned by start_rt_timer

int rt_task_wait_period(void)
  - Suspends execution until the next period
Common Functions (examples)

- BaseP = start_rt_timer(nano2count(1000000));

- RT_TASK* rttask1 = rt_task_init(nam2num("thrd1"), 0, 512, 256);
  *(be careful with the name length)*

- rt_task_make_periodic(rttask1, rt_get_time()+10* BaseP, 20* BaseP );

- rt_task_wait_period();
Generic Task Pseudo-code

start timer

initialize task

make task periodic

while(1) {
    // do task code
    rt_task_wait_period();
}