EE458 - Embedded Systems
Lecture 10 – Intro to Tasks

● Outline
  – Task Definition
  – Task States
  – Task Operations
  – RTEMS Tasks
  – RTEMS Task Directives

● References
  – RTC: Chapter 5
  – CUG: Chapter 5
Lecture 10 – Intro to Tasks
Defining a Task

- Although simple embedded applications may use a super-loop architecture, more complex applications (multiple inputs and outputs, time constraints) must be designed to run multiple tasks concurrently.

- Concurrent design requires the developer to decompose an application into small, schedulable program units. Most RTOS kernels provide task objects and task management services.
Lecture 10 – Intro to Tasks

Defining a Task

• A task is a *schedulable* independent thread of execution. Each task has a name, an ID, a priority, a task control block (TCB) and task code. Task state (CPU registers) are stored in a TCB during a *context switch*.

• The kernel may run its own set of *system tasks* at reserved priority levels. An *idle task* will always be present. Others may include: logging task, exception handling task, debug agent task.
A task may be in either the ready, **blocked** or running states:

- **Ready**: the task cannot run because a higher priority task is running.
- **Blocked**: the task is waiting on a resource or event.
- **Running**: the task has the highest priority and is running.
Finite State Machine of Task States

Figure 5.2 RTC

Task is initialized and enters the finite state machine.

Ready

Task is unblocked but is not the highest-priority task.

Blocked

Task no longer has the highest priority.

Running

Task has the highest priority.

Task is unblocked and is the highest-priority task.

Task is blocked due to a request for an unavailable resource.
Lecture 10 – Intro to Tasks
Typical Task Operations

- A RTOS kernel typically provides task-management services for:
  - Creating, starting and deleting tasks: typically there is a call to first create the task and a second call to start it.
  - Controlling task scheduling: normally there are routines for suspending and resuming tasks, delaying a task, changing priorities, and disabling and enabling the scheduler.
  - Obtaining task information: get the task ID, obtain stack size, etc.
Lecture 10 – Intro to Tasks
Typical Task Structure

- RTOS tasks are usually structured to either run-to-completion or run in an endless-loop:
  - A run-to-completion task is a high priority task that is run at startup to perform initialization (create tasks, semaphores, queues, etc.)
  - The majority of RTOS tasks will run in an endless loop. One or more blocking calls must be made in the loop. (This is necessary so that lower priority tasks will have a chance to run.)
Tasks synchronize and communicate using intertask primitives: semaphores, message queues, signals, pipes, etc. We will discuss these topics during the coming weeks.

We will also discuss how to decompose an application into concurrent tasks (application design).
Lecture 10 – Intro to Tasks

RTEMS Tasks

• In addition to the running, ready, and blocked states, RTEMS defines the following task states:
  - **dormant**: a created task that is not started
  - **non-existent**: uncreated or deleted task

• RTEMS supports 255 task priorities (1 is high, 255 is low).

• In addition, there is no limit to the number of tasks assigned the same priority.
RTEMS Task Modes

• RTEMS tasks have an execution *mode* and *attributes*. By setting the *mode* the following features can be enabled or disabled (the mode can be changed at run-time):
  - preemption (switching to a higher priority task)
  - *timeslicing* (switching to an equal priority task)
  - signal processing (asynch. signaling)
  - interrupt processing (can allow interrupts at a particular level)
By default, preemption, signal processing and all interrupts are enabled while time slicing is disabled.

The set of valid mode constants are listed on the next slide (Section 5.2.5 of the CUG). A mode is specified by ORing mode constants:

`RTEMS_NO_PREEMPT | RTEMS_NO_ASR`

The `RTEMS_DEFAULT_MODES` constant can be used to select the default modes.
RTEMS Task Modes

- RTEMS_PREEMPT - enable preemption (default)
- RTEMS_NO_PREEMPT - disable preemption
- RTEMS_NO_TIMESLICE - disable timeslicing (default)
- RTEMS_TIMESLICE - enable timeslicing
- RTEMS_ASR - enable ASR processing (default)
- RTEMS_NO_ASR - disable ASR processing
- RTEMS_INTERRUPT_LEVEL(0) - enable all interrupts (default)
- RTEMS_INTERRUPT_LEVEL(n) - execute at interrupt level n
Lecture 10 – Intro to Tasks

RTEMS Task Attributes

- Attributes are used to indicate whether or not a task uses the floating point coprocessor and whether it is a local or global task.
- Context switches take longer for tasks that use the coprocessor because the coproc. registers must also be saved in the TCB. (See Sect 5.2.7 for other options.)
- A global task can be contacted by other tasks in a multiprocessor system.
Attributes are set when a task is created and may not be changed. (Task mode settings, on the other hand, can be changed.)

The default attributes indicate that the task is a local task that does not use the floating point coprocessor. The constant RTEMS_DEFAULT_ATTRIBUTES can be used to create a task with default attributes.
Lecture 10 – Intro to Tasks

RTEMS Task Attributes

- The constants that can be ORed to indicate a task attribute are listed below (Section 4.6.5 of the CUG).

  RTEMS_LOCAL  |  RTEMS_FLOATING_POINT

- RTEMS_NO_FLOATING_POINT - does not use coprocessor (default)
- RTEMS_FLOATING_POINT - uses coprocessor
- RTEMS_LOCAL - local task (default)
- RTEMS_GLOBAL - global task
There are directives to create a task, start a task, restart a task, delete a task, suspend a task, resume a suspended task, change a task's priority, change a task's mode, and put a task to sleep for a certain number of ticks or until a particular date and time.

We will briefly look at the most common task directives. Refer to Section 5.4 of the CUG for the complete list of task related directives.
Lecture 10 – Intro to Tasks
RTMS Task Directives

• Tasks are usually created in the `Init()` routine. Here is the `rtems_task_create()` prototype:

```c
rtems_status_code rtems_task_create
(rtems_name           name,
 rtems_task_priority  initial_priority,
 unsigned32           stack_size,
 rtems_mode           initial_modes,
 rtems_attribute      attribute_set,
 Objects_Id           *id );
```
RTEMS Task Directives

Here is an example call:

```c
rtems_id   task_id[2];  // A GLOBAL
rtems_status_code status;
status = rtems_task_create(
    rtems_build_name('T','S','K','1'),
    10,
    RTEMS_MINIMUM_STACK_SIZE,
    RTEMS_DEFAULT_MODES,
    RTEMS_DEFAULT_ATTRIBUTES,
    &task_id[0]
);
```
It is recommended practice to check that the return status from all directive calls is RTEMS_SUCCESSFUL:

A task must be **started** before it will run:

```c
rtems_status_code rtems_task_start(
    Objects_Id       id,
    rtems_task_entry entry_point,
    unsigned32       argument
);
```
A task can be deleted (you can use an ID of RTEMS_SELF to delete the current task):

```c
rttems_status_code
   rtems_task_delete( Objects_Id id );
```

You can restart a task from any state except the **dormant** or non-existant (deleted) states:

```c
rttems_status_code
   rtems_task_restart
       ( Objects_Id id, unsigned32 arg );
```
Lecture 10 – Intro to Tasks
RTEMS Task Directives

- There are directives to suspend and resume a task and to see if a task is suspended:

```c
rtems_status_code
rtems_task_suspend( Objects_Id id );
rtems_status_code
rtems_task_resume( Objects_Id id );
rtems_status_code
rtems_task_is_suspended
( Objects_Id id );
```
Here are a few other task related directives:

```c
rtems_status_code
    rtems_task_set_priority(
        Objects_Id id,
        rtems_task_priority  new_priority,
        rtems_task_priority *old_priority );

rtems_status_code rtems_task_mode(
    rtems_mode  mode_set,
    rtems_mode  mask,
    rtems_mode  *previous_mode_set );

rtems_status_code
    rtems_task_wake_after
    ( rtems_interval  ticks );
```
Lecture 10 – Intro to Tasks
RTEMS Task Directives

- To change the task mode, both the new mode setting and a corresponding mode mask must be given:

```c
rtems_status_code status;
rtems_mode old_mode;
status = rtems_task_mode(
   RTEMS_PREEMPT | RTEMS_NO_TIMESLICE,
   RTEMS_PREEMPT_MASK|RTEMS_TIMESLICE_MASK,
   &old_mode);
```

- A mask of RTEMS_ALL_MODE_MASKS can be used with `old_mode` to reset the mode.