Programming Embedded Systems

Lecture 5 (cont)
Interrupts,
modes of multi-tasking

Monday Feb 6, 2012

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

Different System architectures/ Multi-tasking approaches

Super-loop

- `main` with single big loop containing all functionality of system

```
int main(void) {
    // initialisation
    while (1) {
        // read inputs
        // do computations
        // write outputs
    }
}
```

- Standard method without OS
- Simple, but not very scalable
  → System with multiple tasks?

Fully pre-emptive multi-tasking

- Easy to handle continuous/long-running tasks
- Very responsive programs
- Care required when dealing with critical regions, mutual exclusion
- Standard setup with FreeRTOS
- Context switches can be “expensive” (include systick interrupt)

Cooperative multi-tasking

- Task-switching points have to be specified manually in long-running tasks (FreeRTOS: `taskYIELD`)
- Tasks are also switched out when they are blocked (say, delayed or waiting)
- Enabled in `FreeRTOSConfig.h`

```
#define configUSE_PREEMPTION    0
```

Cooperative multi-tasking (2)

- Simpler than pre-emptive multi-tasking
  - no trouble with critical regions, etc.
  - simple OS kernel
  - potentially more efficient than pre-emptive multi-tasking
- Less responsive than pre-emptive multi-tasking if too few `taskYIELD`

Hybrid multi-tasking

- Cooperative multi-tasking + “manual” pre-emption when necessary (by calling `taskYIELD` from ISR)
- Pre-emption no longer in fixed intervals, only on demand (when event occurs)
- Potentially efficient, but also error-prone

Co-routines (in FreeRTOS)

- Kind of cooperative multi-tasking
- Inactive tasks lose (most of) their state
    - Stack of inactive tasks is removed
    - Local variables lose their values
- Useful for devices with very little RAM, or if *many* tasks need to be created
- http://www.freertos.org/croutine.html
- Similar features in Erlang, Scala