CG2271

Real-Time Operating Systems

Lecture 1

Introduction to RTOS

colintan@nus.edu.sg

NUS National University of Singapore

School of Computing

Learning Objectives

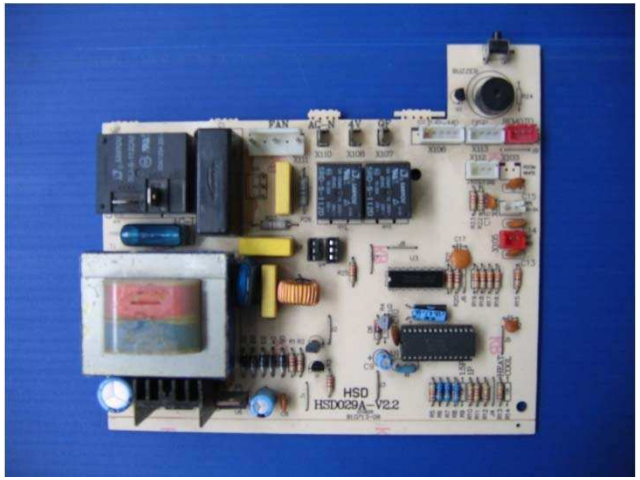
- **By the end of this lecture you should be able to:**
  - Describe what a real-time system is.
  - Understand the issues underlying real-time systems.
  - Know what an operating system is.
  - Appreciate the difference between a desktop/server operating system and a real-time operating system.
- **Why?**
  - These give you an appreciation of why real-time systems are important, and why they require specialized operating systems.

Introduction to RTOS

WHAT ARE REAL-TIME SYSTEMS?

What are Real-Time Systems?

- **"Real time" systems are systems that must react (almost) instantly to inputs.**
  - Examples include:
    - ✓ **Aircraft autopilots.**
    - ✓ **Car engine control units.**
    - ✓ **Satellite navigation systems.**
    - ✓ **Industrial process controllers.**
    - ✓ **Air-conditioning systems.**
    - ✓ **Refrigerators.**
    - ✓ **Car dynamic stability control.**
    - ✓ **…**



What are Real-Time Systems?

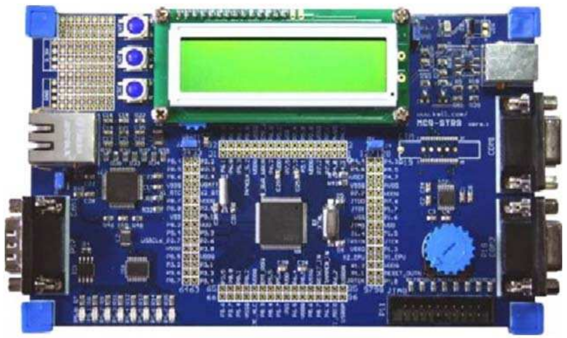
- **Real-time systems are therefore "time critical".**
  - Bad if your air-conditioning system takes 20 minutes to respond to changes in temperature:
    - ✓**You'd either freeze to death or melt to death. Bad.**
  - Extremely bad if your car's anti-lock brakes take 10 seconds to respond!
    - ✓**At 60 km/h, you'd have moved 167m in that time.**

What are Real-Time Systems?

- **There is a spectrum of how time-critical a system is:**
  - Soft Real Time Systems.
    - ✓ **Not particularly time-critical.**
    - ✓ **Failure to meet timing requirements results in inconvenience and degradation of system usability.**
  - Hard Real Time Systems.
    - ✓ **Very important to meet timing deadlines.**
    - ✓ **Failure to meet timing requirements can result in disaster.**

What are Real-Time Systems?

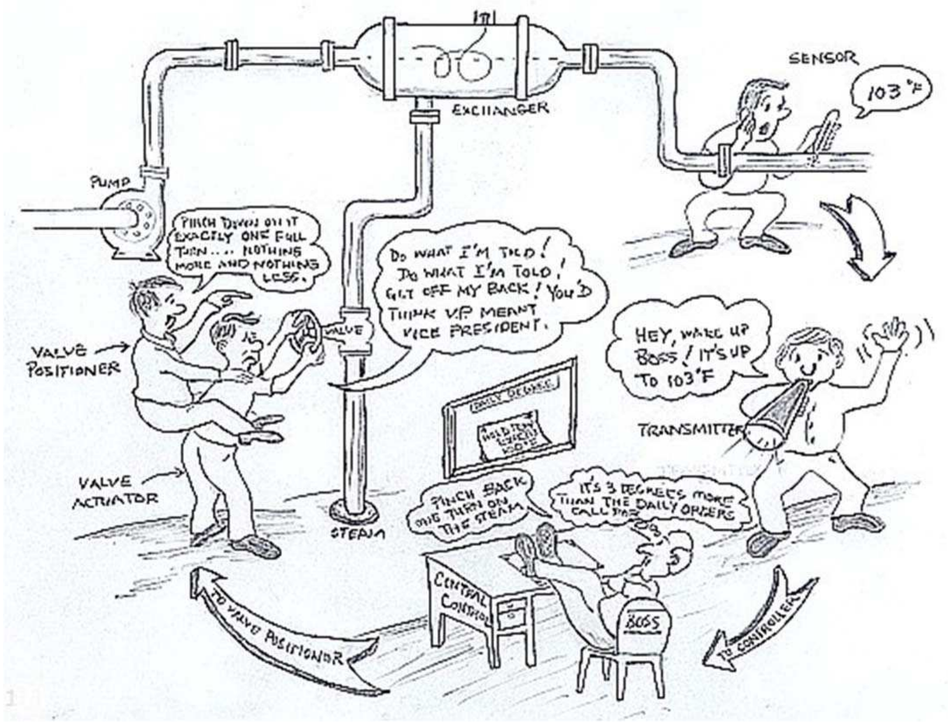
- **Real-time systems also often have limited computing resources.**
  - A high-end configuration (STR91 ARM9 microcontroller)
    - ✓**96KB of RAM**
    - ✓**256KB of Flash**
    - ✓**25 MHz clock speed.**
    - ✓**No hard-disk storage.**



- **Contrast this with a typical PC:**
    - ✓**3.2 GHz processor.**
    - ✓**4 GB of RAM.**
    - ✓**320GB Hard-disk storage.**

What are Real-Time Systems?

- **The focus of real-time systems is also different:**
  - ▪Real-Time Systems
    - ✓**Read sensors.**
    - ✓**Process sensors data.**
    - ✓**Activate actuators to respond to sensor data.**
  - ▪General Computing Systems
    - ✓**More "user oriented".**
    - ✓**Process user inputs to games.**
    - ✓**Drive fancy graphic displays.**
    - ✓**Etc.**



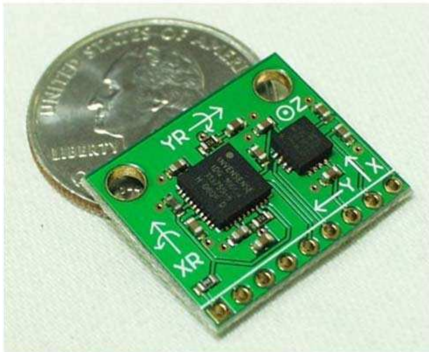
Introduction to RTOS

BASIC TERMINOLOGY

NUS National University of Singapore School of Computing

Basic Terminology: Sensors

- **Sensors:**
  - These are electrical/electromechanical devices that input data into the embedded system.
  - Some examples:
    - ✓**Accelerometers.**
    - ✓**Contact switches.**
    - ✓**GPS receivers.**
    - ✓**…**



  - Sensors are particularly critical as they are often not redundant.
    - ✓**KAL009 shootdown over Soviet Union – possibly caused by failure of inertial-navigation sensors. >200 dead.**

Basic Terminology: Controllers

- **The data from sensors are fed to controllers.**
- **These can be made from:**
  - Complex Programmable Logic Devices – Very simple, very cheap, very limited capability “computers”.
    - ✓**In reality: A collection of flip-flops and gates.**
    - ✓**Used mostly for simpler systems like the lowest-level factory automation systems, washing machine, intelligent refrigerators, etc.**
  - Microprocessors /Microcontrollers– More powerful, more expensive computers.
    - ✓**Used for more demanding applications like autopilots, HDTVs, etc.**

NUS National University of Singapore School of Computing

Basic Terminology: Controllers – "Control Laws"

- **Controllers often rely on "control laws".**
  - These are mathematical descriptions that work out how to reach a target value, based on current sensor readings.
  - E.g. Proportional-Integral-Derivative Control

$$c(k) = K\left(e(k) + \frac{T_c}{T_i}\sum_{n=0}^{t} e(n) + \frac{T_d\,(e(t) - e(t-1))}{T_c} \right)$$

- **Shameless plug: Control law design is covered in detail in CS3271 Software Engineering for Reactive Systems, taught by yours truly. ☺**

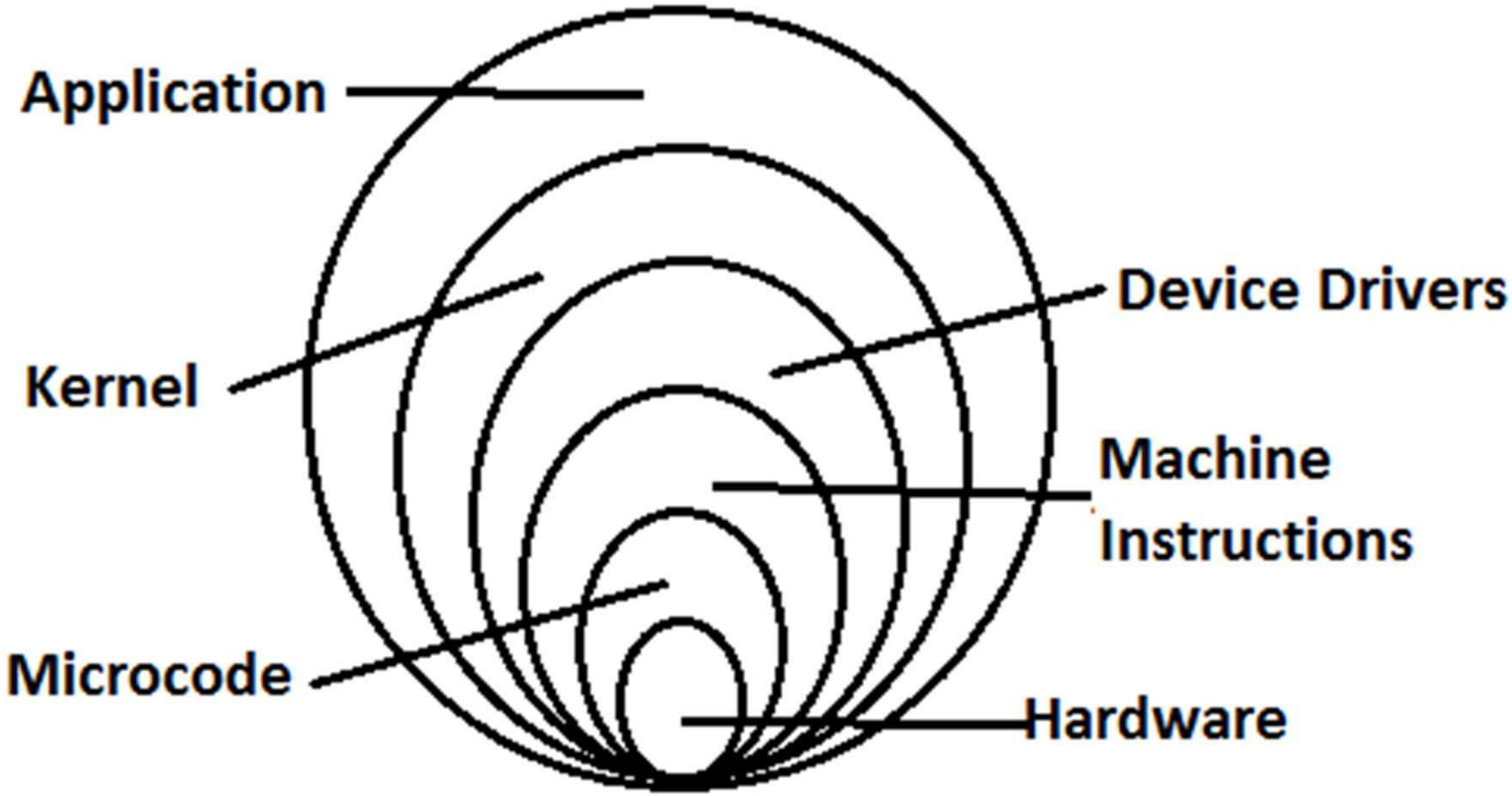
Basic Terminology: Actuators

- **The sensors sense the environment, the controllers compute what to do next. The actuators actually do the work.**
- **Example actuators:**
  - Servos, motors, speakers, tranducers.

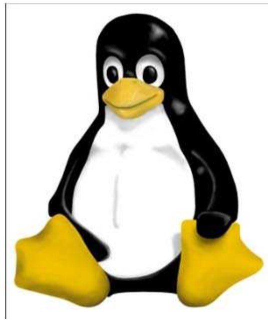
Basic Terminology: Actuators

- **Actuators are very critical as there are often few redundancies:**
  - Air Alaska 261 - 88 dead due to failure of a pitch-control jackscrew.
    - ✓**No redundancy. Plane crashed because a single jackscrew/nut system jammed and then snapped.**
  - American Airlines 585/USAir 427/SilkAir 185 crash – >350 dead in total due to rudder actuator failure.
    - ✓**Actuator failed causing rudder reversal and hard-over, leading to loss of control.**
    - ✓**Similar to turning your car steering wheel slightly to the left, but the car swerves violently to the right.**

Basic Terminology: The Onion Model.



What are Operating Systems?



- **An “operating system” is a suite (i.e. a collection) of specialized “system” software that:**
    - ▪Gives you access to the hardware devices like disk drives, printers, keyboards and monitors.
    - ▪Controls and allocate system resources like memory and processor time.
    - ▪Gives you the tools to customize your and tune your system.
- **Examples include LINUX, MacOS (a variant of LINUX), Windows 7.**

What are Real-Time Operating Systems?

- **Real-time operating systems are OSes that are designed with real-time issues in mind:**
  - Applications are time-critical so schedulers guarantee upper-bounds on execution time.
  - Strong emphasis on reliability.
  - Memory and CPU power are limited, so RTOSs can be made very small and compact – highly customizable.
  - Emphasis on reading sensors and operating actuators rather than interacting with the user.
- **Examples:**
  - RT Linux
  - MicroC/OS-II
  - FreeRTOS

Introduction to RTOS

RTOS APPLICATION OVERVIEW

RTOS: Application Overview

- **We will now look at how an RTOS is used within a real-time system.**
- **There are three important "tasks":**
  - Read the data from the sensors.
  - Compute control responses to these data.
    - ✓**For our application we have two sub-modules: A twin-dynamics module and an adaptive module.**
  - Translate the control responses to actuator outputs.

RTOS: Application Overview

- **We begin by coding the sensor-reading modules using specifications given by the hardware designers.**

Figure 6-14. Multiple Data Byte Write with Internal Address

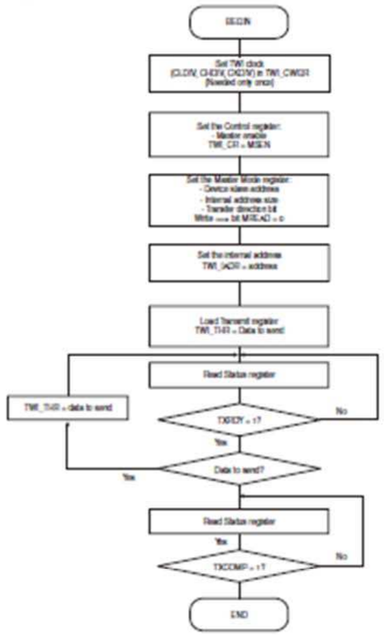


Figure 6-15. Data Write - Slave Receiver Mode

```
uint32_t _len;

void __attribute__((interrupt("IRQ"))) i2c_isr()
{
      uint32_t status=*AT91C_TWI_SR;

      // Send
      if(twi_state==TWI_SEND && (status & AT91C_TWI_TXRDY))
      {

            // If we're done, disable interrupts.
            if(_len==0)
            {
                  *AT91C_TWI_IDR=~0;
                  twi_state=TWI_OK;
            }

            // Send a stop if this is the second last byte.
            if(_len==1)
                  *AT91C_TWI_CR=AT91C_TWI_STOP;

            // Write the next byte
            *AT91C_TWI_THR = *_buffer;
            _len--;
            _buffer++;

      }
```

NUS National University of Singapore School of Computing

RTOS: Application Overview

- **Control responses are specified in the form of "control laws".**
  - PID Control Module:

$$c(k) = K\left(e(k) + \frac{T_c}{T_i}\sum_{n=0}^{t} e(n) + \frac{T_d\,(e(t) - e(t-1))}{T_c} \right)$$

RTOS: Application Overview

- Adaptive Control Module:

$$C_{ij}(t+1) = \begin{cases} \max\{y_i y_j, C_{ij}(t)\} & : \; y_i y_j \geq y_k y_l \;\; \forall\, (1 \leq k, l \leq N) \\ 0 & : C_{ij}(t) < \theta \;\; \forall\, (1 \leq k, l \leq N) \\ \alpha C_{ij} & : \; otherwise \;\; \forall\, (1 \leq k, l \leq N) \end{cases} \qquad (31)$$

$$\Delta w_{bmu} = \varepsilon_{bmu}(v - w_{bmu}) \;\; and \;\; \Delta w_j = \varepsilon_{Nh}(v - w_j) \qquad (32)$$

where

$$Nh(j) = \{i \mid (C_{ji} \neq 0, 1 \leq i \leq N)\}$$

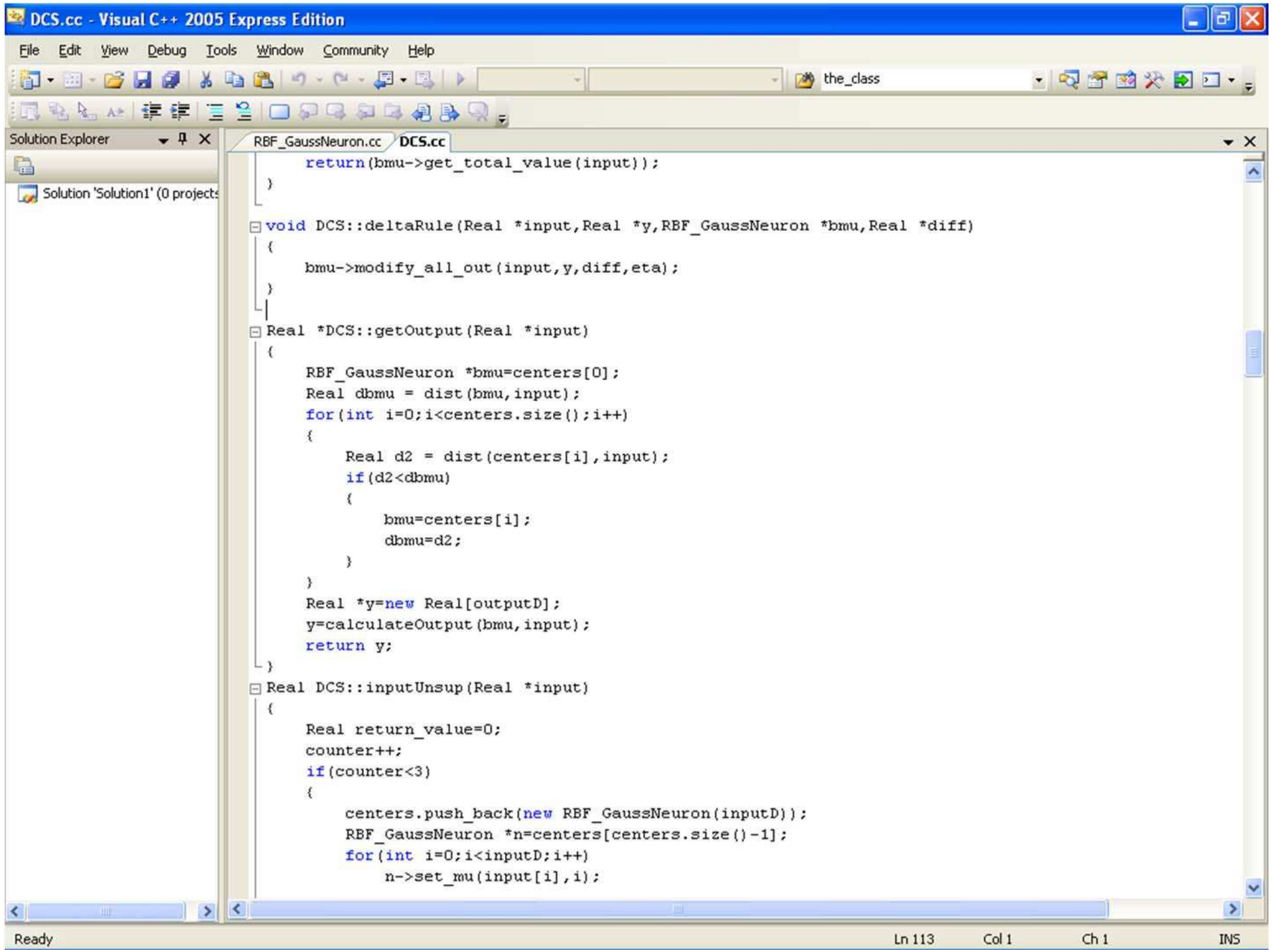
$$c_{bmu} := c_{bmu} + g\left(\frac{\bar{r}}{r_{bmu}}\right) \cdot \varepsilon_{bmu} \cdot (x - c_{bmu})$$

$$\forall u \in Nh(bmu): \qquad c_u := c_u + g\left(\frac{r_u}{r_{bmu}}\right) \cdot \varepsilon_{Nh} \cdot (x - c_u)$$

NUS National University of Singapore
School of Computing

RTOS: Application Overview

- From these, we program the actual code:



RTOS: Application Overview

- **Again we follow the hardware designers' specifications to code the actuator output codes.**

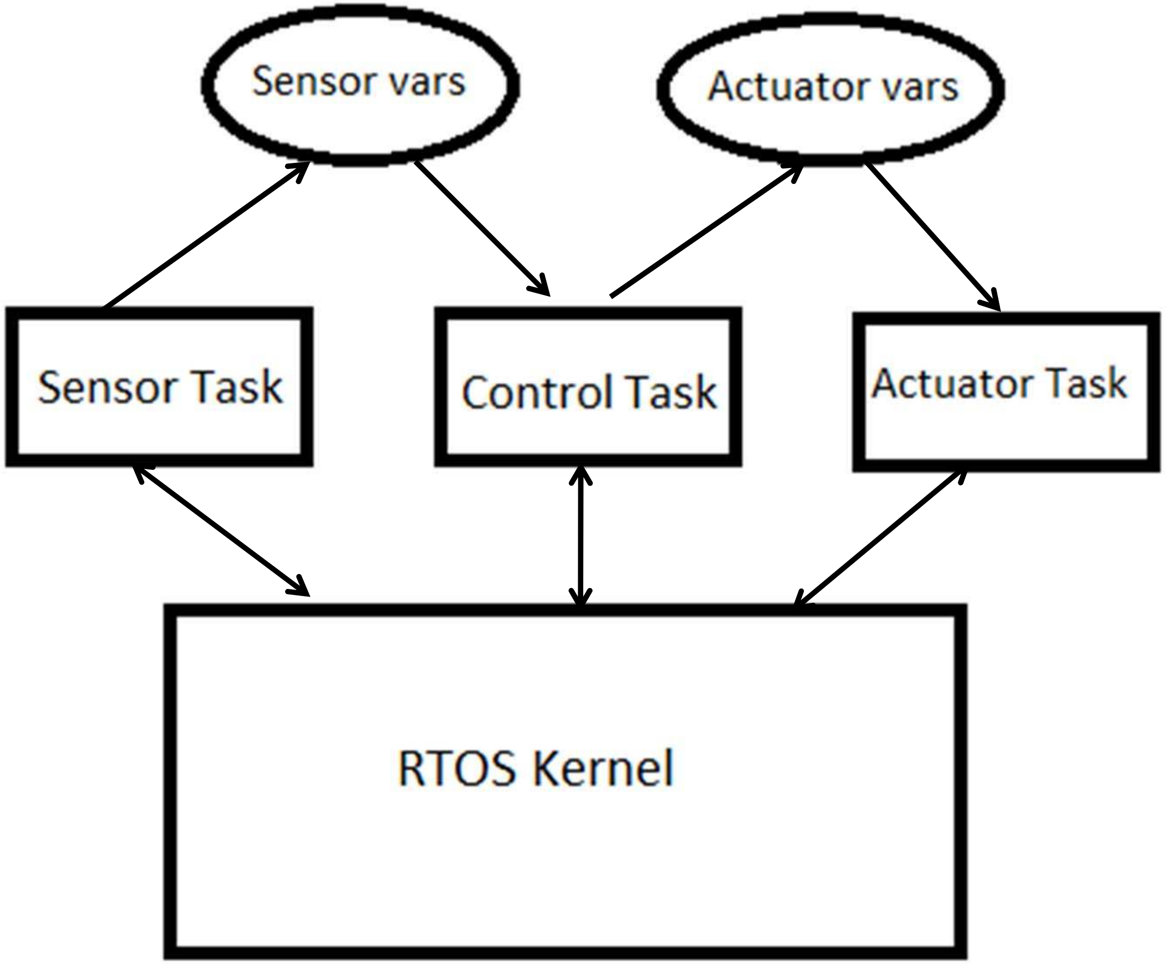
```
if(Cn >= 1000)                                    // Used to limit duty cycle not to have punch through
{
    Cn = 1000;
}
if(Cn <= -1000)
{
    Cn = -1000;
}
if(Cn == 0){                    // Set the speed of the PWM
    DC1B1 = DC1B1 = 0;
    CCPR1L = 0;
}
if(Cn > 0){                     // Motor should go forward and set the duty cycle to Cn
    P1M1 = 0;                   // Motor is going forward
    temp = Cn;
    if(temp^0b00000001){
        DC1B0 = 1;
    }
    else{
        DC1B0 = 0;
    }
    if(temp^0b00000010){
        DC1B1 = 1;
    }
    else{
        DC1B1 = 0;
    }
    CCPR1L = Cn >> 2;           // Used to stop the pendulum from continually going around in a circle
    off_set = off_set +1;       // the offset is use to adjust the angle of the pendulum to slightly
    if(off_set > 55){           // larger than it actually is
        off_set = 55;
    }
```

RTOS: Application Overview

- **Recall that our software portion consists of 3 tasks:**
  - Read the sensors.
  - Compute the control response.
  - Output the control response to the actuators.
- **A RTOS like uC/OS-II can be used to tie these 3 tasks together:**
  - Schedule which tasks to run.
  - Coordinate communications between tasks.
  - Manage interrupts from hardware.
  - …

NUS National University of Singapore
School of Computing

RTOS: Application Overview



RTOS: Application Overview

- **Typically this is what will happen:**
  - The "sensor" task will wait for input from the sensors:
    - ✓**This task is put to sleep – "BLOCKED" – until the sensor comes back with the data.**
    - ✓**When the sensor returns a reading, the "sensor" task will write the value to a "sensor variable" and release a semaphore "S1".**
  - The "control" task will wait for input from the "sensor" task.
    - ✓**It "blocks" on a semaphore "S1" shared with the "sensor" task.**
    - ✓**When the "sensor" task releases that semaphore, the "control" task reads the "sensor variable".**
    - ✓**When it completes computation, it will write its results to a "control" variable and release a semaphore "S2".**

NUS National University of Singapore
School of Computing

RTOS: Application Overview

- The "actuator" task will wait for input from the "control" task.
  - ✓ **It will block on a semaphore "S2" it shares with the "control" task.**
  - ✓ **When the "control" task releases this semaphore, the "actuator" task can read the "control variable" and output to the actuator.**

Summary

- **In this lecture we looked at:**
  - What are real-time systems, and why are they different?
  - Operating systems, and real-time operating systems.
  - An Overview of how an RTOS can be used.