CG2271

Real-Time Operating Systems

Lecture 4

Real-Time Software Architectures

colintan@nus.edu.sg

National University of Singapore

School of Computing

Learning Objectives

- **By the end of this lecture you will be able to:**
  - Understand the various options for designing software for real-time systems.
  - Understand what an RTOS is, and when they are useful.

Introduction

- **A "software architecture" refers to the structure of a piece of software designed to solve a problem.**
  - This structure is important as it determines whether a piece of software can meet the constraints of a problem, and whether it is actually correct.
- **Not all real-time systems require an operating system!!**
  - Some alternatives:
    - ✓**Round-robin.**
    - ✓**Round robin with interrupts.**
    - ✓**Function queue scheduling.**
    - ✓**Timed loops.**

Real-Time Software Architectures

ROUND ROBIN

Real-Time Software Architectures
Round Robin

- **This is the simplest of all architectures. E.g. to process 3 sensors:**
  - 1. Read sensor 1
  - 2. Do something with sensor 1 data
  - 3. Read sensor 2.
  - 4. Do something with sensor 2 data
  - 5. Read sensor 3.
  - 6. Do something with sensor 3 data
  - 7. Goto 1

Real-Time Software Architectures
Round Robin

- **Round Robin architecture works particularly well when:**
  - There are few devices to service.
  - There is no long complicated processing to be done with the data read in.
  - There are no tight time requirements.
    - ✓**E.g. in a multimeter, if a user turns a dial, it is unlikely that he will notice that the loop is currently reading the voltage probes, before it comes round and reads the new dial settings.**

Real-Time Software Architectures
Round Robin

- **However it fails when:**
  - Any device requires attention in less time than it takes for the CPU to go around the loop.
  - If there is lengthy processing to do. In the multimeter example in the book, if it takes 3 seconds to process a voltage reading:
    - ✓ **It may take as long as 3 seconds for the multimeter to respond to the user changing the dial setting.**

Real-Time Software Architectures
Round Robin

- **This architecture is also fragile:**
  - If we added as little as just one more device, the performance may no longer be acceptable.
    - ✓**For example, if the new device takes a long time to get a reading.**

Real-Time Software Architectures

ROUND ROBIN WITH INTERRUPTS

NUS National University of Singapore School of Computing

Real-Time Software Architectures Round Robin with Interrupts

- **In this architecture:**
  - When a sensor has data to send to the CPU, it will interrupt the CPU.
  - The interrupt handler reads the device and sets a flag.
  - The main loop polls these flags, and takes action if any of the flags is set.

Real-Time Software Architectures
Round Robin with Interrupts

- **In the example we are going to see, there are 3 devices that must be read, processed, and the output sent to 3 actuators.**
  - Data from sensors written to 3 global variables dev1_data, dev2_data and dev3_data.
  - We have flags dev1_flag, dev2_flag and dev3_flag to indicate newly arrived data.
  - Devices send interrupts when new data arrives.
  - ISRs are set up to deal with new data from devices.
  - Main loops infinitely checking the dev?_flag.
    - ✓ **If set, process the data in dev?_data.**

Real-Time Software Architectures
Round Robin with Interrupts

```
...
int dev1_flag=0, dev2_flag=0, dev3_flag=0;
int dev1_data=0, dev2_data=0, dev3_data=0;
ISR(DEVICE1_vect)
{
          !! Read device 1 to dev1_data;
          dev1_flag=1;
}
ISR(DEVICE2_vect)
{
          !! Read device 2 to dev2_data;
          dev2_flag=1;
}
ISR(DEVICE3_vect)
{
          !! Read device 3 to dev3_data
          dev3_flag=1;
}
```

Real-Time Software Architectures
Round Robin with Interrupts

```
int main()
{
        !! Set up interrupts, devices, etc.
        dev1_flag=0; dev2_flag=0; dev3_flag=0;
        while(1)
        {
                if(dev1_flag)
                {
                        !! Process dev1_data;
                        !! Write to actuator1;
                        dev1_flag=0;
                } // if
                if(dev2_flag)
                {
                        !! Process dev2_data;
                        !! Write to actuator2
                        dev2_flag=0;
                } //if
```

Real-Time Software Architectures
Round Robin with Interrupts

```
                    if(dev3_flag)
                    {
                              !!Process dev3_data
                              !! Write to actuator 3
                              dev3_flag=0;
                    }  // if
          } // while
} // main
```

- **This architecture is an improvement over simple round robin:**
  - Devices get attended to immediately. Unlikely to suffer loss of data.
- **However, it may still take a while for this data to actually be processed!**

Real-Time Software Architectures Round Robin with Interrupts

- **For example, suppose all 3 devices A, B and C interrupt the CPU:**
  - Data from all 3 devices take 200 ms each in the main loop to process.
  - If the processing sequence is A, B, C, A, B, C, ..., then C will have to wait as long as 400ms to be processed!
- **Even C is a high priority device, it still has to wait for A and B to be done first.**

NUS National University of Singapore
School of Computing

Real-Time Software Architectures
Round Robin with Interrupts

- **Solutions:**
  - Move the processing code for C to its interrupt handler.
    - ✓**Interrupt handler for C will now take 200 ms!**
    - ✓**Unacceptable if the handler is high priority and blocks all lower priority interrupts during this time.**
  - Poll C more often:
    - ✓**Instead of A, B, C, A, B, C,..., we poll A, C, B, C, C, A, C, B, C, C, A, C, B, C, C, ...**

Real-Time Software Architectures

FUNCTION QUEUE SCHEDULING

Real-Time Software Architectures
Function Queue Scheduling

- **This is similar to Round Robin with Interrupts:**
  - Interrupt handlers read the data from the device.
  - BUT instead of setting a flag, it inserts a pointer to the function to process this data.
- **The main loop then takes a function from this queue and executes it.**

NUS National University of Singapore School of Computing

Real-Time Software Architectures
Function Queue Scheduling

```
void function_A()
{
  !! Process data from sensor A
}
void function_B()
{
  !! Process data from sensor B
}
void function_C()
{
  !! Process data from sensor C
}
```

Real-Time Software Architectures
Function Queue Scheduling

```
ISR(DEVICEA_vect)
{
  !! read data from sensor A
  !! Insert pointer to function_A into queue
}
ISR(DEVICEB_vect)
{
  !! read data from sensor B
  !! Insert pointer to function_B into queue
}
ISR(DEVICEC_vect)
{
  !! read data from sensor C
  !! Insert pointer to function_C into queue
}
```

Real-Time Software Architectures
Function Queue Scheduling

```
void main()
{
  while(true)
  {
      !! Remove function from queue and execute
      it.
  } /* While */
}
```

Real-Time Software Architectures
Function Queue Scheduling

- **It is easy to enforce priorities in this scheme:**
  - Just have priorities in the way the queue is managed!
  - For example, function_C is always placed at the front of the queue ahead of everyone else.
  - This gives function_C priority over everyone else.

Real-Time Software Architectures

TIMED LOOPS

Real-Time Software Architectures
Timed Loops

- **A timed-loop is similar to round-robin.**
  - A while loop repeatedly calls functions to handle processing.
  - Each function is called in a fixed order.
- **Difference:**
  - Functions are called at fixed intervals.
  - Before calling the function, the main loop checks if a sufficient number of clock cycles have passed by.
- **This is useful for routines that must be called at fixed times.**

Real-Time Software Architectures
Timed Loops

- **Example:**
  - PID loops require fixed timings for accurate computation.

$$p(t_k) = K(r(t_k) - y(t_k))$$

$$d(t_k) = \frac{T_d}{T_d + Nh}(d(t_{k-1}) - kN(y(t_k) - y(t_{k-1}))$$

$$i(t_{k+1}) = i(t_k) + \frac{Kh}{T_i}(r(t_k) - y(t_k))$$

$$u(t_k) = p(t_k) + i(t_k) + d(t_k)$$

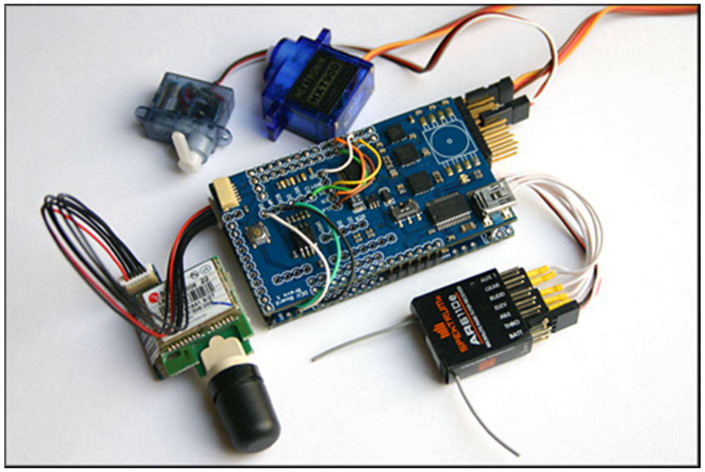
  - The computations for $d(t_k)$ and $i(t_{k+1})$ compute integrations and differentiations.
    - ✓**Accuracy is dependent on period *h*, which must be fixed.**
    - ✓**In our example, we assume a 50 Hz cycle, so *h=20* ms.**

Real-Time Software Architectures
Timed Loops

- **Example code from Ardupilot flight control software.**

```
void loop()
{
        // Execute in 50 Hz loops.
        while(1)
        if (millis()-fast_loopTimer > 19)
        {
            deltaMiliSeconds        = millis() - fast_loopTimer;
            G_Dt = (float)deltaMiliSeconds / 1000.f;
            fast_loopTimer= millis();
                 mainLoop_count++;

            // Execute the fast loop
            // --------------------
            fast_loop();
           // Execute the medium loop
            medium_loop();
        }
}
```



Real-Time Software Architectures
Timed Loops

- **This is how the loop works:**
  - The “millis()” function returns the number of milliseconds since the program started execution.
  - The “`fast_loopTimer= millis();`” statement therefore keeps track of the current time.
  - The “`if (millis()-fast_loopTimer > 19)`” at the start therefore checks that 20ms have elapsed before calling “fast_loop” and “medium_loop”.
    - ✓This works out to a 50Hz cycle.
    - ✓Note that both fast_loop and medium_loop MUST complete under 20 ms or this scheme breaks!

Real-Time Software Architectures
Timed Loops

- **The "fast_loop" function does all the critical flight-control stuff including reading airspeed, computing control responses, etc.**
  - Everything in this loop is executed 50 times a second.

```
void fast_loop()
{
        // This is the fast loop - we want it to execute at 50Hz if possible
        // -------------------------------------------------------------------
        if (deltaMiliSeconds > G_Dt_max)
                G_Dt_max = deltaMiliSeconds;

        // Read radio
        // ----------
        read_radio();
```

Real-Time Software Architectures
Timed Loops

```
// check for throtle failsafe condition
// ------------------------------------
set_failsafe(ch3_failsafe);

// read in the plane's attitude
// ----------------------------
GPS.update();
airspeed  = (float)GPS.airspeed;
calc_airspeed_errors();

// Read in aircraft’s attitude
read_AHRS();
roll_sensor        = -roll_sensor;
pitch_sensor       = -pitch_sensor;
yaw_sensor      = -yaw_sensor;

read_airspeed();
```

Real-Time Software Architectures
Timed Loops

```
        read_AHRS();
        // altitude smoothing
        // ------------------
        if (control_mode != FLY_BY_WIRE_B)
                 calc_altitude_error();

        // custom code/exceptions for flight modes
        // ---------------------------------------
        update_current_flight_mode();

        // apply desired roll, pitch and yaw to the plane
        // ----------------------------------------------

          stabilize();

        // write out the servo PWM values
        // ------------------------------
        set_servos_4();
}
```

Real-Time Software Architectures
Timed Loops

- **The medium_loop function operates at a 10 Hz cycle, but is called 50 times per second by loop().**
  - A counter called "medium_loopCounter" is incremented each time the medium_loop function is called.
  - A switch-case statement is then used to divide the function into 5 chunks, based on the value of medium_loopCounter.
    - ✓**Effectively each chunk is executed only once in every 5 calls to medium_loop.**
    - ✓**Since medium_loop is called 50 times a second, each chunk is called 10 times a second, giving a 10 Hz frequency!**

Real-Time Software Architectures
Timed Loops

```
void medium_loop()
{
        // This is the start of the medium (10 Hz) loop pieces
        // -----------------------------------------
        switch(medium_loopCounter) {

                // This case deals with the GPS
                //-------------------------------
                case 0:
                        medium_loopCounter++;
                        …
                        break;

                // This case performs some navigation computations
                //-------------------------------------------------
                case 1:
                        medium_loopCounter++;
                        …
                        break;
```

Real-Time Software Architectures
Timed Loops

```
                    // unused
                    //------------------------------
                    case 2:
                              medium_loopCounter++;
                              …
                              break;
                    // This case deals with sending high rate telemetry
                    //---------------------------------------------------
                    case 3:
                              medium_loopCounter++;
                              …
                    // This case controls the slow loop
                    //---------------------------------
                    case 4:
                              medium_loopCounter=0;
                              slow_loop();
                              break;
          }
}
```

Real-Time Software Architectures
Timed Loops

- **The last loop is the slow_loop which runs at 3 1/3 Hz.**
  - This is called 10 times per second by medium_loop.
  - We use the same trick as before to "step-down" the frequency.
    - ✓**We have a variable called slow_loopCounter that is incremented each time slow_loop is called.**
    - ✓**A switch-case statement divides slow_loop into 3 chunks.**
    - ✓**Each chunk is executed once every 3 calls, effectively giving a 3 1/3 Hz (10/3) frequency.**

Real-Time Software Architectures

Timed Loops

```
void slow_loop()
{
        // This is the slow (3 1/3 Hz) loop pieces
        //-----------------------------------------
        switch (slow_loopCounter){
                case 0:
                        slow_loopCounter++;
                        !! Handle telemetry comms with ground station.
                        break;
                case 1:
                        slow_loopCounter++;
                        !! Read switches off RC radio
                        !! Read battery level.
                case 2:
                        slow_loopCounter = 0;
                        update_events();
                        break;
        }
}
```

Real-Time Software Architectures

REAL-TIME OPERATING SYSTEMS

Real-Time Software Architectures
Real-Time Operating Systems

- **A Real-Time Operating System (RTOS, pronounced 'arr-toss"), is a suite of software routines that provide:**
  - A (possibly uniform) interface to access hardware devices.
    - ✓**Provides abstraction to layers above the interface.**
    - ✓**Hardware accesses below the interface can be customized to each platform.**
  - Interrupt handling services.
    - ✓**Interrupts from the hardware are vectored to handlers within the RTOS.**
    - ✓**The handlers can be designed so that they respond to different interrupt in *exactly* the same amount of time. This introduces predictability.**

Real-Time Software Architectures Real-Time Operating Systems

- Infrastructure to handle multiple tasks.
  - ✓**Most embedded platforms have single CPUs that can only handle one task at a time.**
  - ✓**The RTOS provides services that automatically switch between tasks, to give the illusion that multiple tasks are executing at the same time.**
- Infrastructure to coordinate tasks.
  - ✓**Message passing, protecting critical sections, etc.**
- Other services.
  - ✓**Memory management.**
  - ✓**Disk access.**
  - ✓**Etc.**

Real-Time Software Architectures
Real-Time Operating Systems

- **When are RTOS good?**
  - Complicated applications that involve many parts that have to interact.
    - ✓**Reading sensors, reading keypads, reading operator buttons, sounding alarms, controlling actuators, sending/receivng data over the network, driving multiple displays, performing computations, etc.**

  +RTOS provides a clean, convenient and predictable way to control complex applications.

  +RTOS are often audited by independent organizations to prove that they are reliable.

    - ✓**E.g. uC/OS is certified by EuroCAE to be safe for installing on commercial airliners.**

  - RTOS are relatively huge, must be customized, and can take some time to learn and understand.

- **The rest of this course is about RTOS, so we won't go into any further detail here.**

Summary

- **In this lecture we looked at various ways to design real-time software.**
  - Round-robin.
    - ✓**Good for simple applications that read/write to a small set of devices with relatively slow cycle times.**
  - Round-robin with Interrupts.
    - ✓**Good for simple applications that use somewhat faster devices, and each device has equal priority.**
  - Function Queue Scheduling
    - ✓**Like round-robin with interrupts, but good when some devices have higher priority than others.**

Summary

- Timed Loops
  - ✓**Like Round Robin, but with device access at fixed intervals, e.g. every 20 ms.**
- Real-Time Operating Systems.
  - ✓**Good for more complex applications with many parts that must be coordinated.**
- **In subsequent lectures we will look at how RTOS work, and how to design applications using an RTOS.**