CS466 Chapter 5

Survey of Software Architectures

Survey of Software Architectures

We will discuss four architectures, starting with the simplest one and then some ideas on which one to choose.

- Round-Robin
- Round-Robin w/interrupts
- Todays Arduino Environment
- Function-Queue scheduling w/interrupts
- Real Time Operating System

Round Robin Architecture

- With a simple round-robin architecture, interrupts are not present. The main loop simply checks each I/O device in turn and services any in need.

```
void main(void)
{
   while(TRUE)
   {
       if(!! I/O Device A needs service)
       {
           !! Take care of I/O Device A
           !! Handle data to or from I/O Device A
       }
       etc. etc.
       if(!! I/O Device Z needs service)
       {
           !! Take care of I/O Device Z
           !! Handle data to or from I/O Device Z
       }
   }
}
```

Round Robin Architecture (cont)

- Round Robin is very simple but it is adequate as long as you can get away with it.
- Use the Digital Multimeter as an example…pseudo code on next slide

Digital Multimeter pseudo code

```
void vDigitalMultiMeterMain(void)
{
    enum{OHMS_1, OHMS_10,…VOLTS_100}eSwitchPosition;
    while(TRUE)
    {
        eSwitchPosition = !! Read the position of the switch
        switch(eSwitchPosition)
        {
            case OHMS_1:
                !! Read hardware to measure ohms
                !! Format Result
                break;
            case OHMS_10:
                !! Read hardware to measure ohms
                !! Format Result
                break;
            case VOLTS_100:
                !! Read hardware to measure volts
                !! Format result
                break;
        }
        !! Write result to display
    }
}
```

Round-Robin comments

- If any one device needs response in less time than it takes the microprocessor to get around the loop in the worst-case scenario, then the system won't work.
- One can squeeze a bit out of the architecture by changing the order of which the devices are tested, but this has limitations because the world is full of devices like pushbuttons which require fairly rapid response.

Round-Robin with interrupts

- A more sophisticated architecture is one in which interrupts are allowed, in order to deal with the need for more rapid response to changing inputs.
- In this scheme, interrupts deal with the very urgent needs of the hardware and then sets a flag; the main loop polls the flags and does the follow-up processing required by the interrupts.
- The interrupt routines provide good response.

Round-Robin w/int pseudo code

```
BOOL fDeviceA = FALSE;
BOOL fDeviceB = FALSE;
…
BOOL fDeviceZ = FALSE;
void interrupt vHandleDeviceA(void)
{
   !!Take care of I/O Device A
   fDeviceA = TRUE;
}
void interrupt vHandleDeviceB(void)
{
   !!Take care of I/O Device B
   fDeviceB = TRUE;
}
void interrupt vHandleDeviceZ(void)
{
   !!Take care of I/O Device Z
   fDeviceZ = TRUE;
}
```

Round-Robin with int/pseudo code

```
void main(void)
{
   if(fDeviceA)
   {
       fDeviceA = FALSE;
       !! Handle data to or from I/O Device A
   }
   if(fDeviceB)
   {
       fDeviceB = FALSE;
       !! Handle data to or from I/O Device B
   }
…
   if(fDeviceZ)
   {
       fDeviceZ = FALSE;
       !! Handle data to or from I/O Device Z
   }
```

Round Robin w/int example

- A simple bridge is used as an example in the text
- The interrupt routines receive characters and write them into queues.
- A sudden burst of characters will not overrun the system, even if encryption and decryption are time-consuming.

Arduino Environment

- Today's popular starter environment for Arduino and lots of other processors is something I'll call the setup/loop pattern.
- You get to fill out the two routines in a C environment
  - `Void setup(void); // get's called once`
  - `Void loop(void); // get called over and over`

Arduino Main

- You are not supposed to mess with the main function which look like…

```
int main(void)
{    _init_internal_();
     setup();
     while (1)
     {
          loop();
     }
}
```

Arduino Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
 */

// Pin 13 has an LED connected on most Arduino boards.
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

More Arduino…

- I've written some moderately complex stuff within the setup/loop environment.
- Like the Round-Robin model, the loop timing must be revisited each time more complexity is added.
- The Arduino project often includes a bunch of periphial control libraries that will save you a boatload of time to:
  - Use directly if your arch is simple.
  - Use indirectly as a source code library sample.
  - Our TivaDriver libraries are a well done example.

Function Queue Scheduling Architecture

- In this architecture, the interrupt routines add function pointers to a queue of function pointers for the main function to call.
- The main routine just reads pointers from the queue and calls the functions.

Function Queue Scheduling pseudo code

```
!!  Queue of function pointers;
void interrupt vHandleDeviceA(void)
{
   Take care of I/O Device A
   Put function_A on queue of function pointers
}
void interrupt vHandleDeviceB(void)
{
   Take care of I/O Device B
   Put function_B on queue of function pointers
}
void main(void)
{
   while(TRUE)
   {
       while(!!Queue of function pointers is empty)
           ;
       !! Call first function on queue
   }
}
```

PSEUDO CODE CONT

```
void function_A(void)
{
   !! Handle actions required by device A
}

void function_B(void)
{
   !! Handle actions required by device B
}
```

Function Queue Scheduling

- This is a worthwhile architecture because there is no rule that says main must call functions in the same order in which the interrupts occurred. It can call them based upon any priority that suits the purposes.
- The worst wait for the highest-priority task code function is the length of the longest of the task code functions plus of course the execution times of any interrupts routines that happen to occur.
- This is a rather better response than the round-robin-with-interrupts response, which is the sum of the times taken by all of the handlers. The tradeoff is that for this better response, is that the response for the lower-priority task function may actually get worse.
- Lower-priority functions may never execute if the interrupt routines schedule the higher-priority functions frequently enough to use up all of the microprocessor's available time.

Real-Time Operating System Architecture

- In this architecture, the most urgent work is taken care of by isr's as before. They then signal that there is work for task code to do. The differences between this and previous ones are:
  - The necessary signaling between the interrupt routines and the task code is handled by the real-time operating system. You need not use shared variables for this purpose.
  - No loops decide what needs to be done next. Code inside the real time operating system decides which of the code functions should run. The real-time o/s knows which of the tasks is the most important to run at a given time.

Real Time O/S

- A side-effect of using the RT O/S is that the system response is relatively stable, even when code is changed.
- The response times for a task function in this architecture generally do not depend upon changing an even lower-priority task. This is not true for round-robin and function queue architectures.

Suggestions for architectures…

- Select the simplest architecture that will meet your requirements
- If your system has response requirements that might require a real time O/S, you should lean toward using a real-time operating system.
- It is possible to create hybrids of the various architectures if it makes sense for your system.
  - For example, a low-priority task could poll certain parts of the hardware that do not need fast response.
  - In a round robin with interrupts, the main loop could poll slower pieces of hw directly rather than reading flags set by isr's.

Summary

- Response requirements most often drive the choice of architecture
- The characteristics of the four architectures discussed are shown in table 15.1…refer to it.
- Generally, you are better off choosing a simpler architecture.
- One advantage of real time systems is that you can buy them and thereby solve some of your problems without having to write the code yourself.
- Hybrid architectures can make sense for some systems.