Embedded Applications

COMP595EA
Chapter 5
Software Architectures

Goal: Control of Response Time

- Software is normally designed to accomplish a task in an efficient manner.
- The primary concern about the design of software in embedded systems is to obtain the greatest amount of control over system response.
  - Systems that require little control and poor response can be done with simple architectures.
  - Rapid response systems will require more complex program design to be successful.

Four Basic Architectures

- We will cover four basic software architectures
  - Round-Robin (simplest)
  - Round-Robin with Interrupts
  - Function-Queue Scheduling
  - Real-time Operating Systems (most complex)

Round-Robin Architecture

- Simplest
- Characterized by the absence of interrupts
- Consists of a main loop that checks each I/O device in turn and services them if needed.
- Cannot suffer from shared data problems
- Latency is limited by the maximum duration of a loop cycle.
- Attractive for simple environments

Example: Multimeter

- The multimeter example
  - very small number of I/O: (switch, display, probes)
  - no particularly lengthy processing (even very simple microprocessors can check switch, take measurement and update display several times per second.)
  - measurements can be taken at any time.
  - display can be written to at any speed
  - small delays in switch position changes will go unnoticed.

Round-Robin Problems

- If any device needs a response in less time than the worst duration of the loop the system won't function.
  - if A and B take 5ms each and Z needs a response time of less than 7ms its not possible.
  - This can be mitigate somewhat by doing (A,Z,B,Z) in a loop instead of (A,B,Z).
    - Scalability of this solution is poor.
  - Even if absolute deadlines do not exist, overall response time may become unacceptably poor.

Problems Cont.

- Round-Robin architecture is fragile
  - Even if the programmer manages to tune the loop sufficiently to provide a functional system a single addition or change can ruin everything.

Round-Robin with Interrupts

- Interrupt routines deal with the very urgent needs of devices
- Interrupt routines set flags to indicate the interrupt happened
- main while loop polls the status of the interrupt flags and does any follow-up processing required by a set flag.

RR with Interrupts Advantages

- More control over priorities.
  - device routines can be serviced in any order
  - processor interrupt priority settings can be used
- Interrupt routines get good response (low latency)
  - Main loop can be suspended
  - Interrupt routines are (must/should be) short.
  - Interrupt code inherently has a higher priority than task code

RR with Interrupts disadvantages

- More complicated than Round-Robin
- Context problems can occur
  - saving and restoring context inside interrupts routines becomes necessary when number of registers/resources is small.
- Shared data problems
  - Debugging becomes more complicated.
- Same latency/priority issues causing (A,C,B,C) still exist.

Function-Queue Scheduling

- Interrupt routines enqueue function pointers for follow-up work onto a queue.
- main routine just dequeues a pointer from the queue and makes a branch to that address.

Advantages

- Latency for high priority devices can be reduced compared to Round-Robin with Interrupts
  - (C,C,C,C,A,C,C,B) is simply a matter of the queuing algorithm.
- In Round-Robin with Interrupts every loop may end up executing every follow-up task.
  - Function-Queue Scheduling guarantees that at most a single follow-up task is executed per loop iteration.

Disadvantages

- Latency for low priority tasks can increase.
- Low priority tasks can actually starve
- Queuing algorithm may be complex/costly to run/code.
- If a low priority follow-up task is very time consuming the latency for higher priority response times will suffer.
  - Once you start processing C you must wait for completion before any other task will get attention.

Real-Time Operating Systems

- Interrupts signal the need for follow-up tasks.
  - But, unlike Function-Queue Scheduling, this is handled by the Real-Time Operating System and not by the interrupt routines manipulating flags or a queue.
- Instead of a loop deciding what to do next the RTOS decides.
- One follow-up task can be suspended by the RTOS in favoring of performing a higher priority task.
  - Biggest difference compares to Function-Queue

RTOS Advantages

- Suspension of tasks allows the worst case wait for the highest priority item to be zero.
- built-in scheduling mechanism yields a system with very stable response characteristics even when changes to the code occur.
- Widely available for purchase.

RTOS Disadvantages

- Can be Costly.
- RTOS are generally complicated and can consume a non-trivial amount of processor cycles.

Architecture Selection

- Select the simplest architecture that will meet your response requirements.
- If your response requirements might necessitate using a real-time operating system then that should probably be your choice.
  - Things rarely get smaller/simpler and its a lot easier to start on a more complicated architecture than to migrate to it later when things grew to hairy.
- If it makes sense create hybrids.