Real Time Operating Systems (RTOS) for ATmega

An alternative to running “on the metal”
Arduino and the Bootloader

• Arduino, the ATmega variant with which we are most familiar, lacks an OS. Rather, a simple program:
  – Waits for a new incoming sketch over serial. If a new sketch is uploaded, the bootloader loads it into flash memory
  – If no new sketch is received, jumps to memory beyond the bootloader (i.e. the current sketch) and executes it
The Big Loop

- Once execution moves past the bootloader, the master loop is entered.
- There is a single “thread” of execution.
- Functions that need to be executed in the background can be implemented via ISRs.
- May be entirely appropriate for certain applications!
  - Ask yourself, “Does the app in question need multiple threads of execution?” Do I really need a RTOS?
RTOS – Advantages

- Built-in support for threads
  - Though the processor only handles one thing at a time, rapid switching gives the illusion of simultaneous execution
- Can result in more efficient use of processor time (assuming efficient scheduling)
- Enables integration of numerous modules. Developers have confidence that threads will be managed efficiently and safely.
RTOS – Advantages

- Threading syntactically more accessible to developers, who may be intimidated / put off by interrupts.

- Formal prioritization of threading (if present) can ensure high priority threads don’t wait on those that are less important.
RTOS – Disadvantages

• Memory Footprint
  – The RTOS and application code for managing threads will, undoubtedly consume extra memory

• Processor Overhead
  – Incurred during thread management, etc.

• Priority Inversion
  – If the RTOS doesn’t have built-in prioritization and a mechanism for enforcing it, a higher-priority thread can find itself waiting for one of lower priority
RTOS – Disadvantages

• Deadlock
  – In any multi-threaded system, the danger of deadlocks arising from resource contention is an issue

• Complexity
  – Coding and debugging can be more difficult when threads are involved

• Learning curve
  – In addition the programming language, OS-specific calls and syntax must be learned
Possible RTOS Uses in Chess Game

• Controller to game piece
  – Whether from controller “bots” or Android handsets, game pieces could have listener threads that wait for movement commands or status inquiries

• Game piece to game piece
  – As game pieces autonomously negotiate the game board, they might keep a “communication channel” thread running in order to converse
Embedded RTOS Examples

• Femto OS
  – http://www.femtoos.org/
  – Small footprint
  – Wide Atmel support

• DuinOS
  – RTOS for Arduino
  – “Native” to Arduino and meant for use in Arduino programming environment
  – Good Code Example using “task loops”:
    • http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1256745982
Embedded RTOS Examples

- **FreeRTOS**
  - Boasts support for numerous devices
    - Support for ATMega seems limited…
  - Threading
    - Unlimited # of tasks
    - Unlimited # of priorities and flexible assignment

- **Nut/OS**
  - Support for numerous ATMega versions
  - Two relevant implementations
    - EtherNut for wired networking
    - BTNut for wireless communication via Bluetooth
Nut/OS Features

• Multi-threading
  – Cooperative (i.e. non-preemptive) multi-threading
    • Therefore, priority inversion shouldn’t be an issue
  – Idle thread is spawned on initialization. Runs concurrently with main routine of application code (which is itself a thread)
  – New threads spawned by call to NutCreateThread
  – Thread synchronization handled by events.
    • Threads can wait on events or wake up other threads by posting events
Nut/OS Features

• In “most cases”, access to shared resources does not require locking.
  – *Hence, no deadlocking in “most cases”?*

• “Deterministic” interrupt latency
  – Responses must occur within strict deadlines

• Modular
  – Only those features needed for the application are included through libraries. Footprint is thereby minimized.
Nut/OS Features

- Driver support for a number of devices, including:
  - Ethernet
  - Serial Flash Memory
  - Multimedia / SD Cards
  - UART Devices
  - I/R Remote Controls
  - Character Displays
  - Direct hardware access and native interrupts are also possible

- Advertised support for:
  - ATmega103
  - ATmega128
  - ATmega2561
  - AT90CAN128
  - Numerous non-Atmega micros, and even the Game Boy Advance
Nut / OS Features

• File Systems
  – UROM
    • Stores files in C arrays that are linked to the application code
  – PHAT
    • Compatible with FAT 12/16/32 file systems
  – UFLASH
    • Optimized for serial flash chips
Nut / OS Features

• TCP/IP Stack

  – Feature rich, containing support for
    DHCP      DNS
    FTP       HTTP
    SMTP      SNMP
    SNTP      syslog

  – Network communication accomplished through familiar syntax, including `fprintf`, `fscanf`, `fgetc`, etc.
Nut/OS Implementation - Ethernut

• “Ethernut is an Open Source Hardware and Software Project for building tiny Embedded Ethernet Devices.”

• 4 Hardware versions

    http://www.ethernut.de/en/hardware/ethernuts.html

• Overview (including Pros and Cons of Nut/OS)

    http://www.ethernut.de/en/
Nut/OS Extension-BTNut

- Extends Nut/OS with Bluetooth stack
- Bluetooth used for communication between sensors in a sensor array
- Website:
  - http://www.btnode.ethz.ch/static_docs/doxygen/btnut/