Low-Level Embedded Programming

Real-Time and Embedded Systems (M)

Lecture 18

UNIVERSITY
of
GLASGOW

Lecture Outline

- Embedded systems programming
  - Interrupt and timer latency
  - Memory issues
    - Protection
    - Virtual memory
    - Allocation, locking, leaks and garbage collection
    - Caches
  - Power, size and performance constraints
  - System longevity
  - Development and debugging
- Example environments

Embedded Systems Programming

- Some real-time embedded systems are complex, implemented on high-performance hardware
- Others must be implemented on hardware chosen to be low cost, low power, light-weight and robust; with performance a distant concern
  - Often-times implemented in C or assembler, fitting within a few kilobytes of memory
  - Correctness a primary concern, efficiency a close second

- How does resource constrained hardware affect applications?

Interrupt and Timed Task Latency

- Key issue in real-time systems is time to respond to events
- Should have predictable worst-case bounds, otherwise cannot reason about the system
  - Both interrupt latency and task scheduling latency

- Examples:
  - Linux has ~600μs typical interrupt handler latency, often runs with 100Hz clock for task scheduling (i.e. 10000μs latency)
    - Long history of problems with system call latency, causing tasks to block for hundreds of milliseconds on certain device accesses
    - Resolved for most common devices, but still unpredictable (and long) latency with uncommon hardware
  - RTLinux claims a maximum 15μs interrupt handler latency, all scheduled tasks execute within at most 35μs of their scheduled time
    - Other hard real-time operating systems offer similar guarantees

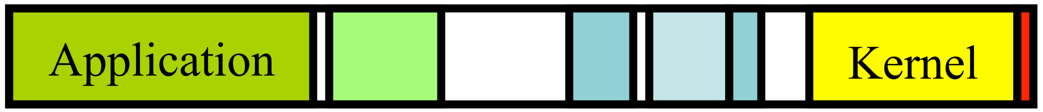
Interrupt and Timed Task Latency

- Why such a difference?
  - Preemptable microkernel, with single address space
    - No context switch, user-to-kernel mode, overhead
  - No virtual memory or memory protection
    - No paging delays
    - No delays while page tables adjusted
  - Device drivers designed with minimal non-preemptable sections
    - Light-weight, prioritised, threads fire in response to interrupts

- Does it matter? It depends on the application…

Memory Protection

- Many embedded systems use a single flat address space
  - Applications, shared libraries, kernel, devices all visible
  - A system or library call is equivalent to a function call



  - Makes system calls, interrupts, very fast and predictable
    - No context switch to kernel mode
    - No adjustment of MMU page tables
  - Consequences
    - No isolation between applications, or between applications and the kernel
    - A change to one part implies that the *entire* system has to be revalidated; difficult as systems become larger
  - Some systems offer limited protection
    - Read only mapping of program/system text; IRQ vectors
    - Optional full memory protection

Memory Protection

- Consequences of offering memory protection:
  - Unpredictable latency
    - May take longer to task switch to/from a protected task
  - Memory overhead
    - Protection provided on a per-page basis, leads to wastage
    - Overhead of maintaining the page tables and protection maps
  - Code overhead
    - Operating system is required to trap illegal access and recover system to a safe state

- Which is easiest: proving the system correct, or writing handlers to safely recover from all possible failures, delays?

Virtual Memory: Address Translation

- Two aspects to virtual memory:
  - Address translation
  - Paging to disk

- Address translation is the act of making a fragmented block of physical memory appear to be a single contiguous block
  - Useful in dynamic systems: enables requests for large blocks of memory to be allocated when there is no physically contiguous block available
  - Adds overhead, since system must manage address translation tables
    - Uses memory, increases context switch time
    - Complicates DMA device access
- Better to pre-allocate static memory pools for real-time tasks
  - Manage the sub-division of address space within the application

Virtual Memory: Paging to Disk

- Disk based virtual memory is supported by many systems that run both real time and non-real time tasks
  - Paging to disk clearly impact real-time performance
  - Unpredictable delays, depending whether page is in memory or on disk

- Systems usually provide ability to (selectively) prevent paging
  - Examples:
    - POSIX allows regions of memory to be locked into RAM and preventing from paging using **`mlock(addr, len)`** and **`mlockall()`**
    - Windows allows all memory owned by a particular thread to be locked
    - LynxOS allows pages of higher priority tasks are locked in memory, but new allocations can page out memory belonging to lower priority tasks

Memory Leaks and Garbage Collection

- An embedded system has to run for a long period of time, without user intervention
- Resource leaks can be problematic:
  - C programs typically have memory leaks due to programmer error
    - Significant problem in long-lived or resource constrained systems
    - Better to pre-allocate static buffers, avoid the chance of a memory leak
    - Be *very* careful to free memory and other resources after use
    - Do you *always* check for out of memory errors? And recover gracefully?
      - Remember the recovery code cannot allocate memory
      - This may include the stack frame needed to make a function call!
  - Modern languages use garbage collection to avoid resource leaks
    - Has a poor reputation due to unpredictable delays when collection occurs
    - However, real-time garbage collection algorithms – with predictable latency, at controlled times – *do* exist
      - Make sure the garbage collector is appropriate for the application

Memory: What is a Small System?

- Embedded systems often very constrained compared to typical desktop computers
  - You may be running on an 8 bit processor, with kilobytes of RAM
  - Operating system typically optimised for the environment, provides only minimal required functions
    - The QNX 4.x microkernel is approximately 12kbytes in size
    - The VRTX microkernel is typically 4-8kbytes in size
  - For comparison:

```
# uname -srm
Linux 2.4.25 i686
# cat tst.c
int main()
{
        return 0;
}
# gcc tst.c -o tst
# ls -l tst
-rwxrwx---  1 csp  csp  4507 Mar 16 00:51 tst
```

Memory: What is a Small System?

- Example: Renesas H8/3217 processor
  - 16 - 60kBytes ROM
  - 512Bytes - 2kBytes SRAM
  - 10-16MHz clock
  - 1 x 16-bit timer; 3 x 8-bit timer
  - 1 x Watchdog timer
  - 2 x UARTS; 2 x I^{2}C interfaces
- The H8/3217 provides a solution to applications where a cost effective solution with up to 4 channels of serial communications is required
  - Monitors, Televisions
  - Radios, Stereo systems
  - Set Top Box system controllers
- The H8/3217 is a member of the H8/300 series of high performance 8/16-bit CPU's. This device is used in applications where a high level of communications capability is required
- The combination of 2 high speed UARTS capable of transmitting data asynchronously at 500k baud and two channels of I^{2}C capable of transmitting at over 400k bits per second make this devices a powerful communications processor

[Adapted from Renesas website]

Effects of Cache

- You may be running on a more modern processor
  - PowerPC 405CR embedded processor
    - 32 bit RISC processor, compatible with desktop PowerPC
    - 133MHz or 266MHz clock speed
    - 500mW power consumption
      - Compare: Pentium M ("Centrino") processor consumes up to 24.5W
    - CodePack^{TM} compression of executables
    - Likely has several megabytes of memory
    - [Various types of PowerPC and ARM processors commonly used]
  - Relatively cheap, comparatively high performance, low power

- Has a small cache, which you may want to disable:
  - Processor and memory speeds are closely matched
    - Compare to desktop processor, with order magnitude difference
  - Simpler to predict memory access times without the cache
  - Cache improves average response times, but introduces unpredictability

Power, Size and Performance Constraints

- Embedded systems often battery powered or power sensitive
  - What influences power consumption?
    - Power consumption $\propto$ (clock speed)^2
    - Memory size and processor utilization
- May have to be physically small and/or robust
- May have strict heat production limits
- May have strict cost constraints
  - That processor is slower, but 10¢ cheaper, the production run is 1 million, you paid your salary for the next couple of years…

- Used to throwing hardware at a problem, and writing inefficient – but easy to implement – software
  - Software engineering based around programmer productivity
  - The constraints may be different in the embedded world…

System Longevity

- Embedded systems often safety critical or difficult to upgrade

Medical devices	CD or DVD player
Automotive or flight control	Washing machine
Railway signalling	Microwave oven
Industrial machinery	Pacemaker

- May need to run for several years, in environment where failures either kill people, or are expensive to fix
  - Can you guarantee your system will run for 10 years without crashing?
  - Do you check all the return codes and handle all errors?
  - Fail gracefully?

Development and Debugging

- Embedded systems typically too limited to run a compiler
- Develop using a cross compiler running on a PC, download code using a serial line, or by burning a flash ROM and installing

- Often limited debugging facilities:
  - Serial line connection to host PC
  - LEDs on the development board
  - Logic analyser or other hardware test equipment

Example Environments

- VxWorks
- QNX
- Symbian

VxWorks

- Monolithic kernel; POSIX with real time extensions
- Proprietary APIs to control more advanced features
  - Message queues with timeouts
  - Control of priority inheritance on semaphores
  - User processes can enable/disable interrupts
- Defaults to a single address space, with address translation
  - Processes can request memory protection, if desired
  - Processes can control which regions of memory are cached
- Focus on hard real time, deeply embedded systems
  - Runs on the Mars rovers, Pathfinder
    - Pathfinder had problems due to uncontrolled priority inversion causing some tasks to miss their deadlines; caused system to repeatedly reset to safe state
    - Enough debugging code left in that the problem could be resolved, and new code uploaded

QNX

- Pure microkernel system
  - Many optional components, scales from 12kbytes to run on high end SMP machines with gigabytes of memory
- Native support for threads with a single address space
  - Memory protection optional
- Message passing abstraction for inter-task communication
  - Very efficient, due to single address space
  - Tasks inherit priority of the messages
  - Messages can be blocking, variable sized, or fixed size non-blocking
- Network stack, TCP/IP
- Full GUI, web browsers, Java, etc

- Focus on real time embedded, but user-facing, systems

Psion/Symbian

- Example: Psion series 5mx – precursor to Symbian mobile phones
  - 16M RAM, 16M ROM
  - 36MHz ARM710 processor
  - Preemptive multitasking, GUI, C++
  - Software: agenda, word processor, spreadsheet, address book, email, web browser, calculator, jotter, sketch, voice notes, Java
  - Runs for ~1 month on 2 AA batteries
  - Reliable: runs for years without rebooting…
    - Small, efficient, power-aware and robust code

- Focus on telephony and soft real time systems
  - Often run under a hard real time OS using a two-level scheduler

Summary

By now you should…

- Be thinking about the system issues, and how features that improve general purpose performance hinder real time jobs
- Be thinking about the constraints on embedded systems, and differences in how they are engineered
- Know a little about different systems that are available

Tomorrow: summary and overview of the module