ECE 471 – Embedded Systems
Lecture 13

Vince Weaver

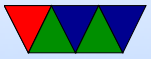
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

18 October 2016

Announcements

- How is HW#6 going?

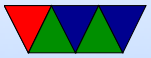


Midterm Review

1. Embedded Systems
   Note, "full OS running" does not mean it's not an embedded system
   Digress on supercomputer

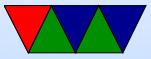
2. Operating Systems

   (a) Benefit of OS: abstraction, portability
   user friendly? easier to program? libraries? maybe self hosted?



(b) Drawback of OS: overhead, not all features available

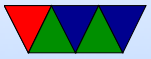
(c) Themostat: can you write an OS in 8kb? low power– not necessarily. more expensive to have OS: why?



Homework 5 Review

- C Coding Hints:
  Use the pre-processor (Pound defines) when odd constants are involved. Code is much easier if you have something like `HT16K33_REGISTER_DIMMING` than `0xE0`.

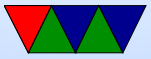
- Error checking redux.
  If can't handle an error, exit, don't just print a message and charge through.
  I only expect reasonable error checking. In theory every



printf can fail, but people don't check for error on every call. Important things like file opening and memory allocations you must check and do something.

- 7a. GPU is used on boot on Pi. This is unusual, for the GPU to handle the booting. It's *not* unusual for an embedded board to have a GPU. It's not necessarily unusual to have off-chip SD-card storage.

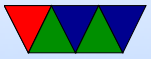
- 7b. fat32 is used primarily because it is simple and widely used, as well as mostly patent free. Simple enough for firmware/bootloader to read (CPU itself cannot do it



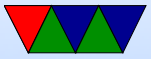
in hardware, there is some software involved). You could use implement it yourself in a HW assignment (well, enough to load one file from the root dir). Not so much NTFS or HFSplus or ext4, let alone btrfs. fat/fat12/fat16/vfat/fat32/exfat
"manages space effectively" – no

- 7c. What I was looking for was the bootloader, which is the chunk of code responsible for loading the kernel into memory.

- 7d. Why not scan? Because they are reserved.



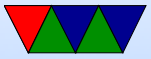
- Command line? diff and wc?



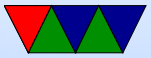
Context Switching

- OS provides the illusion of single-user system despite many processes running, by switching between them quickly.

- Switch rate in general 100Hz to 1000Hz, but can vary (and is configurable under Linux). Faster has high overhead but better responsiveness (guis, etc). Slower not good for interactive workloads but better for long-running batch jobs.

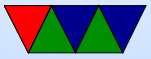


- You need to save register state. Can be slow, especially with lots of registers.
- When does context switch happen? Periodic timer interrupt. Certain syscalls (yield, sleep) when a process gives up its timeslice. When waiting on I/O
- Who decided who gets to run next? The scheduler.
- The scheduler is complex.
- Fair scheduling? If two users each have a process, who runs when? If one has 99 and one has 1, which runs



next?

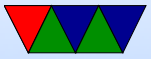
- Linux scheduler was O(N). Then O(1). Now O(log N). Why not O(N^3)



Real Time Constraints

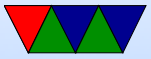
What are real time constraints?

- Time deadlines that hardware needs to respond in.
- Goal not performance, but response time



Types of Real Time Constraints

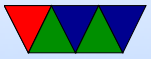
- Hard – miss deadline, total failure (people die?)
  Antilock brakes?
- Firm – result no longer useful after deadline missed
  lost frames in video, missed frames in video game
- Soft – results gradually less useful as deadline passes.
  Caps lock LED coming on?



Constraints depend on the Application

Can almost always come up with a scenario where a soft constraint could become hard.

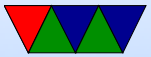
For example: Unlocking a car door taking an extra second? Not hard real-time, except maybe if your car is about to crash and you need to escape quickly.



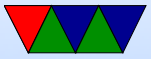
What can cause problems with real-time?

Sources of "Jitter"

- Interrupts. Taking too long to run; being disabled (cli)
- Unpredictable nature of modern CPUs. Caches, branch-predictors, etc.
- Operating system. Scheduler. Context-switching.
- Dynamic memory allocation, garbage collection.

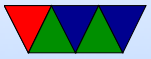


- Slow/unpredictable hardware (hard disks, network access)



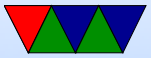
Determining worst case behavior.

- Hard on modern processors. Easier on stm32l than on raspberry pi running Linux
- STM32L is in-order. Program in assembly. Turn off interrupts. You know exactly when everything is happening.
- Pi, with OS. Can you disable interrupts?



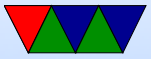
Common OS strategies

- Event driven – have priorities, highest priority pre-empts lower
- Time sharing – only switch at regular clock time, round-robin

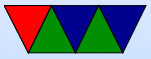


Scheduler example

- Static: Rate Monotonic Scheduling – shortest job goes first
- Dynamic: Earliest deadline first
- Three tasks come in. a. finish in 10s, 4 long. b. finish in 3, 2 long, c. finish in 5, 1 long
- In order they arrive, aaaabbccc bad for everyone
- RMS: cbbbaaaa works

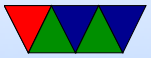


- EDF: bbbcaaaa also works.
- Lots of information on various scheduling algorithms



Priority Inversion Example

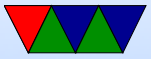
- Task priority 3 takes lock on some piece of hardware
- Task 2 fires up and pre-empts task 3
- Task 1 fires up and pre-empts task 1, but it needs same HW as task 3. Waits for it. It will never get free.
- Space probes have had issues due to this.



Real Time OS

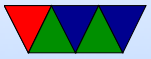
Who uses realtime?

- Timing critical situations. Cars, medical equipment, space probes, etc.
- Industrial automation. SCADA. Stuxnet.
- Musicians, important to have low-latency when recording
- High-speed trading



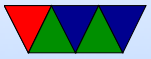
PREEMPT Kernel

- Linux PREEMPT_RT
- Faster response times
- Remove all unbounded latencies
- Change locks and interrupt threads to be pre-emptible



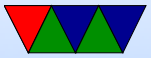
Linux PREEMPT Kernel

- What latencies can you get? 10-30us on some x86 machines
- Depends on firmware; SMI interrupts (secret system mode, can't be blocked, emulate USB and like)' Slow hardware; CPU frequency scaling; nohz
- Special patches, recompile kernel
- mlockall() memory in, start threads and touch at beginning, avoid all causes of pagefaults.



Co-operative real-time Linux

- Xenomai
- Linux run as side process, sort of like hypervisor



Other RTOSes

- Vxworks
- Neutrino
- Free RTOS
- Windows CE