ECE 471 – Embedded Systems
Lecture 19

Vince Weaver

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

vincent.weaver@maine.edu

16 October 2017

Announcements

- Project handout posted to website

Project Preview

- Can work in groups
- Embedded system (any type, not just Pi)
- Written in any language (asm, C, python, C++, Java, etc.)
- Do some manner of input and some manner of output using the various capabilities we discussed
- I have a large amount of i2c, spi, and other devices that you can borrow if you want to try anything interesting.
- Past projects: games, robots, weather stations, motor

controllers, music visualization, etc.

- Will be a final writeup, and then a short presentation and demo in front of the class during last week of classes.
- Can compliment another project, but must have some original code

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

Locking

- When shared hardware/software and more than one thing might access at once
- Multicore: thread 1 read temperature, write to temperature variable

  thread 2 read temperature variable to write to display

  let's say it's writing 3 digit ASCII. Goes from 79 to 80. Will you always get 79 or 80? Can you get 70 or 89?
- How do you protect this? With a lock. Special data structure, allows only one access to piece of memory,

others have to wait.

- Can this happen on single core? Yes, what about interrupts.
- Implemented with special instructions, in assembly language
- Usually you will use a library, like pthreads
- mutex/spinlock
- Atomicity

Priority Inversion Example

- Task priority 3 takes lock on some piece of hardware (camera for picture)

- 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. (camera for navigation?)

- Space probes have had issues due to this.

Real Time Operating System

- Can it be hard real time?

- Simple ones can be mathematically provable

- Otherwise, it's a best effort

Priority Based, like Vxworks

- Each task has priority 0 (high) to 255 (low)
- When task launched, highest priority gets to run
- Other tasks only get to run when higher is finished or yields
- What if multiple of same priority? Then go round-robin or similar

Is Regular Linux a RTOS

- Not really
- Can do priorities ("nice") but the default ones are not RT.

Real Time Linux

- Project to have a small supervisor RTOS and run Linux as a process
- Code that needed a compatible OS interface could call into this process-Linux, but it could always be pre-empted
- Not supported anymore?

PREEMPT Kernel

- Linux PREEMPT_RT
- Faster response times
- Remove all unbounded latencies
- Change locks and interrupt threads to be pre-emptible
- Have been gradually merging changes upstream

Typical kernel, when can you pre-empt

- When user code running
- When a system call or interrupt happens
- When kernel code blocks on mutex (lock) or voluntarily yields
- If a high priority task wants to run, and the kernel is running, it might be hundreds of milliseconds before you get to run

- Pre-empt patch makes it so almost any part of kernel can be stopped (pre-empted). Also moves interrupt routines into pre-emptible kernel threads.