# ECE 598 – Advanced Operating Systems
# Lecture 20

Vince Weaver

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

vincent.weaver@maine.edu

12 April 2018

## Announcements

- Project topics were due
- HW#10 might be delayed

## Project Notes

- Unknown interrupt errors caused by power-sag, noticeable when stressing processor(s)
- Varying levels of project difficulty.
- If you get stuck, don't stress too much.
- Also understand that the OS we're using is very limited, so for something like decoding a filesystem just getting files dumped from it is an accomplishment, let alone hooking up all the hooks in the OS.
- Old fashioned PS/2 keyboard. IBM PS/2, not

playstation. PC standard for keyboards until USB took over. Much simpler than USB, mostly just a serial port, sort of vaguely like SPI. The keyboard has a micrcontroller that does the scanning. Doesn't return ASCII, but scancodes which you might have to decode (more fun, different countries keyboards return different scancodes for letters)

- Writing C programs in userspace. Hard to use the subset that can be handled (imagine if doing C++). gcc is too clever, will do things like map

```
printf("%s","Hello World");
```

```
to
puts("Hello World");
```

which is trouble if we don't implement puts.

## Multi-Processing

## Hardware Concerns

- Multi-processing
  Symmetric, Asymmetric
  SMP vs CMP (Symmetric and Chip Multi-processing)
- Multi-threading
  (Hyperthreading, SMT)
- Shared memory vs Distributed
  Shared memory, a CPU can write a value to memory, read it back and it will be different (another CPU can

write to it)

- UMA, NUMA, CC-NUMA (cache-coherent)
  Non-uniform memory access
- How many copies of the OS? One per core or single image? One per core is more like a cluster.

## Multi-Processor Resource Sharing

- How are resources shared in SMP system?
- Any core can access any of the devices. Need locking.
- What about interrupts?
  - Have one core handle all interrupts?
    Might have better cache behavior
  - Round-robin interrupts to each core?
    Reduces load on core0 but hurts others.

  - Balance interrupt load across processors?

## OS Support for SMP

- How can we have multiple cores share one OS-image?
- Big-kernel-lock, but poor performance
- Only parts of OS happen at once. Scheduler can run at same time as serial driver or filesystem read or page fault
- Split up with fine-grained critical sections.
- Suddenly deadlocks are a problem.
- What kinds of locks?
  - Spinning easiest, but poor performance.

- Switch threads. Multi-threading OS?
- Linux has kernel threads (look in top for things starting with k or rcu). Interrupt handlers have fast handler and worker threads.

## SMP Scheduling

- 4 processors, 5 jobs
  How to avoid ping-ponging? Better to make two processes slow or all of them?
- Gang scheduling – if you have processes that are using IPC (or multithreads) you want to schedule all at the same time so can communicate without having to wait through multiple context switches.
- Keeping jobs on same CPU started on (why is this good?) Cache behavior. TLB, NUMA.

Why might you want to move them?

- When might you want to run everything on one core even though lots available? Power! Can put rest of CPUs to sleep.
- How do you online/offline hotswap processors.

## Initializing Multicore on Raspberry Pi

- Bare Metal
  - Detect which processor you are on

```
mrc p15, 0, r3, c0, c0, 5
ands    r3, #3                    /* CPU ID is Bits 0..1 */
bne wait_forever              /* If not CPU zero, go to sleep */
```

  - “park” the extra CPUs. Put in tight loop, wfe (wait for exception) when wake, check a flag to see if they should start and jump to address if true. Otherwise, back to sleeping.
  - To wake, use SEV to send event

- Raspberry Pi booth firmware does this for you
  It copies some code to 0x0 and executes it before jumping to your code at 0x8000
  - This code parks the other cores
  - each process has a mailbox, if you write an address there it will jump to it core 1: 0x4000009C core 2: 0x400000AC core 3: 0x400000BC
  - They are waiting in WFE so have to send SEV too
- Other things you will need to do:
  - Set up stacks for each CPU (why can't they all share the CPU0 stack?)

- Start up virtual memory and caches
  Locking depends on the caches working
- Start them into idle thread
- Start scheduling jobs?

## Multicore Concerns

## Race Conditions

- Shared counter address
  RMW on ARM
  Thread A reads value into reg
  Context switch happens
  Thread B reads value into reg, increments, writes out
  Context switch back to A
  increments value, writes out
  What happened?
  What should value be?

## Critical Sections

- Want mutual exclusion, only one can access structure at once

1. no two processes can be inside critical section at once
2. no assumption can be made about speed of CPU
3. no process not in critical section may block other processes
4. no process should wait forever

## How to avoid

- Disable interrupts. Heavy handed, only works on single-core machines.
- Locks/mutex/semaphore

## Mutex

- mutex_lock: if unlocked (0), then it sets lock and returns if locked, returns 1, does not enter.
  what do we do if locked? Busy wait? (spinlock) re-schedule (yield)?

- mutex_unlock: sets variable to zero

## Semaphore

- Up/Down
- Wait in queue
- Blocking
- As lock frees, the job waiting is woken up

## Locking Primitives

- fetch and add (bus lock for multiple cores), xadd (x86)
- test and set (atomically test value and set to 1)
- test and test and set
- compare-and-swap
  - Atomic swap instruction SWP (ARM before v6, deprecated)
  - x86 CMPXCHG
  - Does both load and store in one instruction!
  - Why bad? Longer interrupt latency (can't interrupt

    atomic op)
  - Especially bad in multi-core
- load-link/store conditional
  - Load a value from memory
  - Later a store to same memory address.
  - Only succeeds if no other stores to that memory location in interim
  - ldrex/strex (ARMv6 and later)
- Transactional Memory

## Locking Primitives

- can be shown to be equivalent
- how swap works:
  - lock is 0 (free). r1=1; swap r1,lock
  - now r1=0 (was free), lock=1 (in use)
  - lock is 1 (not-free). r1=1, swap r1,lock
  - now r1=1 (not-free), lock still==1 (in use)

## Memory Barriers

- Not a lock, but might be needed when doing locking
- Modern out-of-order processors can execute loads or stores out-of-order
- What happens a load or store bypasses a lock instruction?
- Processor Memory Ordering Models, not fun
- Technically on BCM2835 we need a memory barrier any time we switch between I/O blocks (i.e. from serial to GPIO, etc.) according to documentation, otherwise loads could return out of order

## Resources

- If you do not give exclusive access, bad things can happen. Imagine one process printing a document, half done and another task switched in and also starts writing to the printer.
- Pre-emptible resource
- Non-preemptible resource.
- Usually protected by locks.
- More complex if protected by two or more locks (need two resources)

## Deadlock

- Two processes both waiting for the other to finish, get stuck
- One possibility is a bad combination of locks, program gets stuck
- P1 takes Lock A. P2 takes Lock B. P1 then tries to take lock B and P2 tries to take Lock A.

## Livelock

- Processes change state, but still no forward progress.
- Two people trying to avoid each other in a hall.
- Can be harder to detect

## Starvation

- Not really a deadlock, but if there's a minor amount of unfairness in the locking mechanism one process might get "starved" (i.e. never get a chance to run) even though the other processes are properly taking and freeing the locks.

## How to avoid Deadlock

- Don't write buggy code
- Reboot the system
- Kill off stuck processes
- Pre-emption (let one of the stuck processes run anyway)
- Rollback (checkpoint occasionally)

## Priority Inversion

- Low-importance task interrupts a high-priority one
- Say you have a camera. Low-priority job takes lock to take picture.
- High-priority task wants to use the camera, spins waiting for it to be free. But since it is high-priority, the low priority task can never run to free the lock.

## Locking in your OS

- When?
- Interrupts
- Multi-processor
- Pre-emptive kernel (used for lower latencies)
- Big-kernel lock? Fine-grained locking? Transactional memory?
- Semaphores? Mutexes
- Linux futexes?

## Where does our OS need locks?

- Does a single-core operating system need locks?
  Yes, interrupts can cause similar troubles
- Any shared resources
- What if multiple processes try to write the console at the same time?
- What if try to update the memory allocation/free list at same time?