ECE 598 – Advanced Operating Systems
Lecture 24

Vince Weaver

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

vincent.weaver@maine.edu

26 April 2016

Announcements

- I will post a HW#10 that is short answer, practice for the second midterm, with solution.
- There will not be a final coding project. Was hoping to have a multi-tasking one but the assembly language is hard and you will be busy with your projects.
- Don't forget project presentations, there should be time for everyone to present on May 5th

HW#9 Review

- Font printing not a problem?
  Didn't cover transparency
- Gradient – example code. Tricky with division and such when not multiple of 256
- Something cool. `marie.fnt` is only an 8x8 font as well as Tolkien Elf Runes which is why it does weird things
- Alignment: align on boundary
  Why on GPU? For the interface we need 4-bits to pass

channel number, so we need a pointer where the bottom 4 bits are zero

Midterm Review

- Benefit/Downside to using an operating system
- Virtual Memory
  One operating system feature made easier by VM?
- Filesystem
  pseudo-filsystems
  reasons to use filesystems
  filesystem features: holes, etc.
- Graphics

Why is the GPU in the Pi unusual? Why is it aligned?

- Scheduling
  Scheduler should be fast
  Round-robin is simplest
- Threading
  User vs Kernel threads
- Multiprocessors/Locking

Multi-Processing

Hardware Concerns

- Multi-processing
  Symmetric, Asymmetric
  SMP vs CMP (Symmetric and Chip Multi-processing)

- Multi-threading
  (Hyperthreading, SMT)

- Bus (small amounts) – for memory just puts request on the bus. If busy it waits. Why can this be bad if large

CPUs?
Cache – each CPU has local cache. Have to keep cache coherent though. Large (¿16?) traditional cache coherence doesn't scale well. Then use crossbars, switching networks. Gets more complex.

- 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 SMP on ARM

- Detecting the processors
- Need to power them up
- Then need to somehow (implementation dependent) set the PC for each
- Typically leave them waiting in WFE (similar to WFI but also will wait for SEV event). SEV sends event to all cores waking any in WFE state.

- On x86 IPI (inter-processor interrupts) are used during bringup

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 store instruction 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?

Does our OS need locks?

- We don't have many shared resources yet.
- Setting/reading the time, if not-atomic and updated by interrupt
- What if multiple processes try to write the console at the same time?

IPC – Inter-Process Communication

- Processes want to communicate with each other. Examples?
- Two issues:
  getting the message across
  synchronizing
- signals
- network, message passing (send, receive)

- shared memory (mmap)

Linux

- Signals and Signal handlers
  Very much like interrupts
  Concurrency issues much like threading

- Pipes
  stdout of one program to stdin of another
  one-way (half duplex)
  ls — sort
  pipe system call / dup

C library has popen()

- FIFOs (named pipes)
  exist as file on filesystem
- SystemV IPC
  shared memory, semaphores ipcs
- Just use mmap
- Unix domain sockets
  Can send file descriptors across

- Splice – move data from fd to pipe w/o a copy? VM magic?
- Sendfile. zero copy?

IPC in the news

- kdbus – dbus into kernel to make it faster
  Desktop Bus, D-Bus
  allows communication on a desktop bus between apps and kernel
  example – incoming skype call can notify system, and apps like the audio adjust and mp3 player can pause music and set up microphone
  multicast
  example – battery low notification, apps can listen, save

state, prepare to shut down.

- Kernel developers resist it
  Most because who has to maintain it?
  Also how well designed is it? can it be used by other tools?
  We already have a lot of IPC in the kernel, can it be made generic?
  It is faster, but what if user programs just bloat to negate this?

- Worse is better / the right thing

- New Jersey vs MIT
- Is it better to spend lots of time coming up with a perfect specification on paper, then implement it?
- Is quicker/easier to understand better than complex and perfect?
- Should you come up with something “good enough” and let it grow naturally?