Review

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Exam will be closed-book

- Who is reading comp.risks?

- About today's review

  – Mentioning key concepts

  – No promise of exhaustive coverage

  – Reading *some* of the textbook is advisable!

- Will attempt a 3-slide summary at end

OS Overview

- Abstraction/obstruction layer
- Virtualization
- Protected sharing/controlled interference

Hardware

- Inside the box – bridges
- User registers and other registers
- Fairy tales about system calls
- Kinds of memory, system-wide picture
  - User vs. kernel
  - Code, data, stack
  - Per-process kernel stack
- Device driver, interrupt vector, masking interrupts

Hardware

- DMA
- System clock
  - "Time of day" clock (aka "calendar")
  - Countdown timer

Memory hierarchy

- Not covered in class yet, not on exam

Process

- Pseudo-machine (registers, memory, I/O)
- Life cycle: fork()/exec()
  - specifying memory, registers, I/O, kernel state
  - the non-magic of stack setup (argv[])
  - the non-magic function that calls main()
- States: running, runnable, sleeping
  - also forking, zombie
- Process cleanup: why, what

Thread

- Core concept: schedulable set of registers
  - With access to some resources ("task", in Mach terminology)
  - Thread stack
- Why threads?
  - Cheap context switch
  - Cheap access to shared resources
  - Responsiveness
  - Multiprocessors

Thread types

- Internal
  - optional library
  - register save/restore (incl. stack swap)

- Features
  - only one outstanding system call
  - "cooperative" scheduling might not be
  - no win on multiprocessors

Thread types

- Kernel threads

  – resources (memory, ...) shared & reference-counted

  – kernel manages: registers, kstack, scheduling

- Features

  – good on multiprocessors

  – may be "heavyweight"

Thread types

- M:N
  - M user threads share N kernel threads
    - dedicated or shared

- Features
  - Best of both worlds
  - Or maybe worst of both worlds

Thread cancellation

- Asynchronous/immediate
  - Don't try this at home
  - How to garbage collect???

- Deferred
  - Requires checking or cancellation points

Thread-specific data

- printf("Client machine is %s\n", thread_var(0));
- reserved register or stack trick

Race conditions

- Lots of "++x vs. --x" examples using table format

- The setuid shell script attack

  - (as an example in a different arena)

Wacky memory

- Memory writes may be re-ordered or coalesced
- That's not a bug, it's a feature!

Atomic sequences

- short

- require non-interference

- typically nobody *is* interfering

- store->cash += 50;

- "mutex" / "latch"

Voluntary de-scheduling

- "Are we there yet?"

- We *want* somebody else to have our CPU

- *Not-running* is an OS service!

- Atomic:

    – release state-guarding mutex

    – go to sleep

- "condition variable"

Critical section problem

- Three goals
  - Mutual exclusion
  - Progress – choosing time must be bounded
  - Bounded waiting – choosing cannot be unboundedly unfair

- Lecture 7
  - "Taking Turns When Necessary" algorithm
  - Bakery algorithm

Mutex implementation

- Hardware flavors
    - XCHG, Test&Set
    - Load-linked, store-conditional
    - i860 magic lock bit
    - Basically isomorphic
- Lamport's algorithm *(not on test!!!)*
- "Passing the buck" to the OS (or why not!)
- Kernel-assisted instruction sequences

Bounded waiting

- One algorithm discussed
- How critical in real life?
  - Why or why not?

Environment matters

- Spin-wait on a uniprocessor????

- How reasonable is your scheduler?

  – Maybe bounded waiting is free?

Condition variables

- Why we want them

- How to use them

- What's inside?

- The "atomic sleep" problem

Semaphores

- Concept
  - Thread-safe integer
  - wait()/P()
  - signal()/V()

- Use
  - Can be mutexes or condition variables

- 42 flavors
  - Binary, non-blocking, counting/recursive

Monitor

- Concept
  - Collection of procedures
  - Block of shared state
  - Compiler-provided synchronization code
- Condition variables (again)

Deadlock

- Definition
  - N processes
  - Everybody waiting for somebody else
- Four requirements
- Process/Resource graphs
- Dining Philosophers example

Prevention

- Four Ways To Forgiveness
- One of them actually commonly used

Avoidance

- Keep system in "safe" states
  - States with an "exit strategy"
    - Assume some process will complete, release resources
    - Make sure this enables another to finish, etc.
    - Banker's Algorithm

Detection

- Don't be paranoid (but don't be oblivious)
- Scan for cycles
  - When?
  - What to do when you find one?

Starvation

- Always a danger
  - Understand vs. deadlock
- Solutions probably application-specific

Context switch

- yield() by hand (user-space threads)

  – No magic!

- yield() in the kernel

  – Built on the magic process_switch()

  – Inside the *non-magic* process_switch()

    - Scheduling
    - Saving
    - Restoring

- Clock interrupts, I/O completion

Scheduling

- CPU-burst behavior
  - "Exponential" fall-off in burst length
  - CPU-bound vs. I/O-bound
- Preemptive scheduling
  - Clock, I/O completion
- Scheduler vs. "Dispatcher"
- Scheduling algorithm criteria
  - Who cares about what?

Scheduling – Algorithms

- FCFS, SJF, Priority
- Round-robin
- Multi-level
- Multi-processor (AMP, SMP)
- Real-time (hard, soft)
- The Mars Pathfinder story
  - priority-inheritance locks

Memory Management

- Where addresses come from
  - Program counter
  - Stack pointer
  - Random registers
- Image file vs. Memory image
- What a link editor does
  - relocation
- Logical vs. physical addresses

Swapping / Contiguous Allocation

- Swapping

    – Stun a process, write it out to disk

    – Memory can be used by another process

- Contiguous allocation

    – Need a big-enough place to swap in to

    – External fragmentation (vs. internal)

Paging

- Fine-grained map from virtual to physical

  – Page address $\Rightarrow$ frame address

- Page table per process

  – Per-page bits: valid, permissions, dirty, referenced

  – Fancy data structures

    - Multi-level page table
    - Inverted page table
    - Hashed/clustered page table

Segmentation

- Concept
  - Hardware expression of "memory region"
  - Protection boundary, sharing boundary
- Typically combined with paging
  - The beautiful complex x86
    - Useful summaries (for P3 if nothing else)
      - Text pp. 309-311
      - "x86 Segmentation for the 15-410 Student" on web site

Less is more

- Software-managed TLB
  - Choose *your own* page table structure
  - "Embody" it via behavior of TLB miss handler

Virtual Memory

- Observations
  - Some stuff is *"never"* needed in memory
  - Some stuff isn't needed in memory *to start*
  - Some stuff is *sometimes* needed in memory

- Approach
  - RAM is just a *cache* of system memory
  - Page-valid bits record swapping out of pages
  - Page-fault handler fixes everything up

Page-fault handling

- Map address to region

- Deduce *semantic* reason for fault

- Special techniques

  – COW

  – Zero pages

  – Memory-mapped files

Paging

- Page replacement policy
  - FIFO, optimal, LRU
  - *Reality: LRU approximations*
    - Clock-like algorithms
- Gritty implementation details
  - Backing store policy
  - Page buffering
  - Reclaim faults

Paging

- Frame allocation policy
  - Equal/proportional/...
- Thrashing
  - Just not enough pages
  - Working-set model
  - Fault-frequency model
- Reducing paging
  - Simple program optimizations

Summary – What is an OS?

- Parts of a machine
  - Memory, registers
  - Interrupts/traps and their handlers
- Parts of a process (incl. thread)
  - Memory, registers
  - System calls (stubs, handlers)

Summary – What is an OS?

- How to assemble machine parts into process parts
  - How to make virtual memory from physical memory
  - How to make a process from memory and registers
    - And an executable file
- How to share a machine among processes
  - (and how to share a process among threads)
  - Context switch/yield

Summary – Synchronization

- Basic RAM-based algorithms

  – Be able to read one and think about it

- Mutex, condition variable

  – When to use each one, and why

  – What's inside each one, and why

Summary – Deadlock

- A fundamental OS problem
  - Affects every OS
  - No "silver bullet"
- What you need for deadlock
- Prevention, Avoidance, Detection/Recovery
  - What each is, how they relate