CS140 – Operating Systems Midterm Review

Feb. 5th, 2009

Derrick Isaacson

Midterm Quiz

- Tues. Feb. 10th
- In class (4:15-5:30 Skilling)
- Open book, open notes (closed laptop)
  - Bring printouts
  - You won't have time to learn the material but they will likely help as a quick reference
- Will cover first 10 lectures (through 2/5)

Outline

1. OS Overview
2. Processes & threads
3. Concurrency
4. Synchronization
5. Scheduling
6. Advanced scheduling
7. Linking
8. Virtual memory HW
9. Virtual memory OS
10. Memory allocation

OS Overview

- OSes make hardware useful to programmer
- Useful interface
  - System calls
- Protection
  - Resource allocation
  - Preemption – allows OS to regain control
  - Memory protection – protect one process' memory from another process' bad actions
- Properties to consider
  - Skew – temporal/spatial locality
  - Fairness vs. Throughput

Processes & Threads - kernel view

- Data for a process is stored in a Process Control Block (PCB) – think of "struct thread" from projects
- Includes?
  - Page directory - defines its virtual address space
  - Saved registers
  - Priority
  - Open fd's
  - State (runnable, exiting, …)

Processes & Threads - threads

- From lecture: "A thread is a schedulable execution context"
- Kernel threads – pros & cons?
  - create/join are system calls - ~10x slower than function call
  - Still has big data structures used for processes
  - Can more easily take advantage of SMP
- User threads
  - More lightweight
  - More flexible
  - Thread API just function calls
  - Hard to take advantage of SMP
  - Can deadlock even if one thread blocks on another

Concurrency

- Sequential Consistency
  - Maintain *program order* on individual processor
  - Ensure *write atomicity*
  - Can use memory barrier to preserve observable program order
  - Most of the concurrency techniques we discussed assumed sequential consistency
- Why would disabling interrupts be good?
  - May be most efficient method on uniprocessor
- What do you need for a multiprocessor?
  - HW support such as test_and_set/xchg that gives you atomic read/write

Synchronization - Deadlock

- Given limited resources A, B, C, D – can I have deadlock in the following situations? Why/why not?
  - I always acquire resources in alphabetical order
    - No - no circularity in request graph
  - An "older" thread can steal a resource from a "younger" one that holds it
    - Yes, if 2 threads can have same timestamp
    - No if timestamps unique – we have preemption
  - I order all resources at startup
    - No – no hold and wait

Scheduling

- Worst case workloads for each algorithm
- FCFS
  - CPU bound job will hold proc and no I/O work done (convoy effect), long job arriving just before short ones - increases avg time to completion
- SJF
  - Long I/O job keeps getting CPU ahead of short jobs
- RR
  - Multiple jobs of same size
- BSD (what are cons)
  - Absolute priorities, can't transfer, inflexible, many knobs to tweak

Advanced Scheduling

- Lottery scheduling (tickets = chance of getting CPU)
  - What does the scheduler need to do when a process dies to adjust number of tickets?
    - Just reduce total count of tickets in system.
  - What kind of an application would not work well?
    - Multimedia, anything that needs a predictable latency
- Stride scheduling (tickets, stride, pass)
  - What does it fix over lottery?
    - Reduces average error
  - What is a pathological case
    - Bad response time for 101 procs w/ allocations 100:1:1:...:1
- BVT (effective virtual time, weight, warp factor)
  - What are it's goals
    - Provide "universal" scheduler including for real-time and interactive processes

Linking

- During which pass of the linker would the following message be generated (Win 06 midterm)?
  - "External Reference FOO not found."
  - Second pass

Virtual Memory HW

- What are pros and cons of segmentation (base & bounds)
  - Pros: easy, makes data relocatable
  - Cons: fragmentation, not transparent to program
- What kind of fragmentation do you get
  - With segmentation?
    - external
  - With paging?
    - internal

Virtual Memory OS

- What is the clock algorithm used for?
  - Page replacement
- What does it approximate?
  - LRU
- If you have 8 GB of memory what could go wrong with the clock algorithm?
  - There are 2 million pages. By the time you get around then almost all of them will likely be accessed. This gives a poor approximation of LRU.
- What can you do to fix it?
  - Add a second hand that clears accessed bits ahead of the page selecting hand.

Memory Allocation

- If you're implementing distributed shared memory using mprotect/sigaction, what protection level do you give the following types of pages:
  - Ones that only you're caching
    - R/W
  - Ones that you and others are caching
    - R/O
  - Ones that others are writing to
    - No access (invalid)

Questions?