Feedback

Dave Eckhardt
de0u@andrew.cmu.edu

Homework Q2

- Was deadlock prevention an option for P2?
- Answer
  - We couldn't violate the “mutual exclusion” deadlock requirement
    - “because the data structures would have been messed up”
- “Pixie dust” theory
  - Sprinkle mutex_lock() powder on sick programs
  - More is better
- Tape drive != “struct mutex”

Homework Q4

- In the IBM System/370, memory protection is provided through the use of keys.
- A key is a 4-bit quantity.
  - How many 4-bit keys are there?
  - One per frame?
  - One per process?
  - One per process memory page?

Homework Q4

- Each 2 KB block of memory has a key (the storage key) associated with it.
- The CPU also has a key (the protection key) associated with it.
- A store operation is allowed only if both keys are equal, or if either is zero.
  - Can a process change its protection key?
  - Does the kernel need protection_key=0 to write to kernel memory?

Homework Q4

- (a) Explain why an operating system might ***typically*** arrange for memory pages allocated to a single user process to have ***different*** storage keys.
- ***Three*** issues
  - Storage key for page 1
  - Storage key for page 2
  - ***Protection key***

Homework Q4

- Assume pk = 1
  - Some pages have sk = 1
    - The process can write to them
  - Some pages have sk != 1
    - The process can't write to them
- What is the ***common case*** of non-writeable pages?

Homework Q4

- (b) Explain why an operating system kernel might be designed so most kernel code would ***not*** execute with the protection key equal to zero.
- Ok...
  - pk != 0, so pick one, i.e., pk = 2
  - Some pages have sk = 2 (which ones?)
  - Some pages have sk != 2 (which ones?)
  - When ***is*** pk = 0 used?

Exam – overall

- Grade distribution
  - 24 A's (90..100)
  - 20 B's (80..89)
  - 12 C"s (70..79)
  - 4 other
- No obvious need to curve
- Final exam could be harder
- Grade change requests: end of week

Exam - overall

- “And then the OS ...”

Exam – overall

- “And then the OS ...”
  - This is an ***OS class***!
  - We are ***under the hood***!
  - The job is to understand the parts of the OS
    - What they do
    - How they interact
    - Why

Q1

- Are keyboard interrupts really necessary?
- Same
  - Input may arrive early (input queue)
  - Processes may arrive early (waiting queue)
- Focus on what is ***different***
  - ***Detecting*** new input
  - ***Carrying it to*** existing input queue/wait queue

Q1

- “Polling” approach
  - When?
  - How long?
- “Process” approach
  - When?
  - How long?
    - Eating every other quantum is ***not good***
  - How to interact with wait queue?

Q2 (a)

- The “process exit” question
- Sum of process memory is 256 K
- Memory freed on exit is 50 K
  - Not a multiple of 4 K
    - (so not an x86, no big deal)
  - ***Not*** “approximately” 16 K stack + 32 K heap

Q2 (b)

- Process state graph
- Went well overall

Q2 (c)

- Explain why you have no hope of accessing memory belonging to your partner's processes.
- Key concept: ***address space***
  - Everybody gets ***their own*** 0..4 GB
- Other options possible
  - Segmented address space (Multics)
    - But you needed to explain
    - Common case: every main() in same place
  - Sparse virutal address space (EROS)

Q3: load_linked()/store_conditional()

- ***Required*** to consider multi-processor target
  - test-and-yield() is bad
    - unless you carefully explained it
- Common concern: lock/unlock conflict
  - Real load-linked()/store-conditional() a bit better
  - Still an issue (see Hennessey & Patterson)
    - random back-off
    - ***occasional*** yield

Q4: “Concentration” card game

- “Global mutex” approach
  - “Solves” concurrency problems by ***removing*** concurrency!
  - Can be ***devastating***
    - (not a technique we covered in class)
- Deadlock avoidance/detection approaches
  - Hard to get right
  - There ***is*** another option

Deadlock ***prevention***

- “Pass a law”
  - So ***every possible sequence*** violates one of:
    - Mutual exclusion
    - Hold & Wait
    - Non-preemption
    - Wait cycles

Common case

- Violate “wait cycles”
- Establish ***locking order***
  - ***Total order*** on mutexes in system
  - Pre-sort locks according to order
  - Or, dump & start over
- Good locking order: memory addresses
  - &card[i][j]
    - each lock is unique
    - every lock is comparable to every other lock

A novel solution

- One mutex per ***card pair***
  - 36 cards, (36*35)/2 = 630 mutexes
- Can make sense for ***small n***
  - Lamport's fast mutual exclusion algorithm
    - (related approach)

A subtle mistake

```
i1 = generate_random(0, 5);
j1 = generate_random(0, 5);
i2 = generate_random(i1, 5);
j2 = generate_random(j1, 5);
```

- Good news
  - No wait cycles
- Bad news?

Q5: Critical Section Protocol

- “Hyman's algorithm”
  - Comments on a Problem in Concurrent Programming
    - CACM 9:1 (1966)
    - (retracted)
- Doesn't provide mutual exclusion
- Doesn't provide bounded waiting

Q5: Critical Section Protocol

- You should understand these problems
- You won't implement mutexes often
- ***Thought patterns*** matter for concurrent programming