Exam Feedback

Dave Eckhardt
de0u@andrew.cmu.edu

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 ...”
  - ***No!***

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?
- Conditions which remain the 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?
- "Create a special process" approach
  - When does it run?
  - How long?
    - Polling for all of every other quantum is ***not good***
  - How to interact with wait queue?

Q1 (summary)

- Observe that CPU quantum can be set to 5 ms
- Observe people don't need echo for 50 ms
- Re-wire scheduler
  - Scan keyboard hardware for new scan codes
  - Invoke pseudo-interrupt
    - Basically, same code as your keyboard interrupt handler
- Pseudo-interrupt gives keystroke to process
  - Put near front of scheduler 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? We didn't say it's an x86...
  - Trying to change the problem:
    - 50K “is approximately 16K stack + 32K heap”

Q2 (a) - summary

- Virtual-freed != physical-freed due to ***sharing***
- Could be copy-on-write
- Could be shared read-only text regions
- Insight: physical memory is ***used*** to make virtual
  - They are not “the same”

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 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?
  - ***Starvation*** of certain cards
    - (well, ***serious*** bias against)

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