15-410

"..."I'll be reasonable as soon as I get everything I want"..."

Exam #1
Oct. 19, 2005

Dave Eckhardt

Synchronization

Checkpoint 2 – Monday, in cluster

- Reminder: context switch ≠ interrupt
  - Later other things will invoke it too

A Word on the Final Exam

Disclaimer

- Past performance is not a guarantee of future results

The course will change

- Up to now: “basics”
  - What you need for Project 3
- Coming: advanced topics
  - Design issues
  - Things you won't experience via implemention

Examination will change to match

- More design questions
- Some things you won't have implemented (text useful)

Outline

Question 1

Question 2

Question 3

Question 4

Q1 – kernel_main()

```
int kernel_main(void) { return(kernel_main()); }
```

What does it do?

- Base: “unrestrained stack growth”
- Key: *then* what?

Good

- Stomp code, LDT, device regs; no-mem machine-check

Ok, depending

- Segmentation fault

Not as good, can be ok

- Page fault

Q1 – kernel_main()

Avoid

- “kernel will kill you”
- “scheduler will run somebody else”
- “you will starve other processes of memory
- P1 ⇒ “There...is...no...pilot!” (Laurie Anderson)
  - Those things happen for P3 only if you can arrange it...

Also avoid

- It is like an exec(), it is like a fork(), ...

Q2 – Thread-based “simulation”

```
void make_object_thread(int id, char *name) {
  obj_desc_t desc = { id, name };
  thr_create(object_thread, (void*)&desc);
}
void *object_thread(void *arg) {
  obj_desc_t *desc = (obj_desc_t *) arg;
  printf(“...”, desc->object_id,
    desc->owner_name);
}
```

Q2 – Thread-based “simulation”

Key concepts

- Dangling reference to expired stack frame
- Race condition

Common misconceptions

- “Mistaken” array-size computation

```
char *owners[] = { “Mike”, “Rahul” };
const int n_owners = sizeof (owners) /
  sizeof (owners[0]); /* yep */
```

- When stack frame is “gone” access will fault
  - What do we mean by “gone”?

Q2 – Thread-based "simulation"

Approaches

- Serialize! (Run one thread to finish, then run next)
  - Then they're procedure calls, not threads!
- Big global array, tell new thread its index
  - Fine for exam question, maybe not great for 10^6 objects
- Baton-passing
  - main loop acquires {semaphore,mutex}
  - new thread releases it once bits are copied

  ⇒ synchronization hand-off as part of "every" create not ideal
- creator malloc()/new-thread free()
  - You may get *some* contention on malloc() mutex, but you can expect it to be less

Q2 – Thread-based “simulation”

Grading note

- Don't “prove too much”
  - Many “explanations” of seg fault could “prove” every seg fault
- Best answers explained both odd output and seg fault

Q3 – Calvin & Hobbes

“Calvin-o-tron” cookie management system

Key concepts (clearly mention both)

- That linked-list code is both right and wrong
  - It's *called* “queue”, but it *implements* “stack”
- Stacks are double-plus un-fair
  - Can be *infinitely* unfair – key word: “starvation”

Note

- A large Unix vendor shipped kernel-provided semaphores based on a stack.
- How could they not notice????
  - Well...it always worked ok for them... (how?)

Q4 – sys_write() / “superbuffers”

Key concept

- Not the best plan for success:
  - “No matter what” loop around mutex_lock()
    - » “I don't want the world...I just want your half” --TMBG

Approaches

- Just Serialize!
  - Only one thread in superbufferacquire() at once
    - » Deadlock can *always* be solved by serialization
    - » But: bufferacquire() really does take a long time
    - » Multi-processor PCs are no longer rare
    - » Generally, your manager won't be impressed

Q4 – sys_write() / “superbuffers”

Approaches

- “As available”
  - Lock as many buffers as we can right now, opportunistically
  - Problem
    - » All systems get busy
    - » Busy time is a bad time to enter inefficient mode
    - » Some systems are *always* busy
- Try all-at-once allocation, else yield(-1)
  - This is the recipe for ... ?
- “Apply standard avoidance algorithm”
  - Pretty costly hammer for this case...something is special

Q4 – sys_write() / “superbuffers”

Observation

- Buffer use isn't indefinite / random
- Once you have your 8 you'll proceed to release all
- It'll always be 8 (a known fraction of all the buffers)

Plan

- Split allocation/locking apart from store-back I/O
- Allocate 8 at once
  - Use a “who chooses next” queue to provide fairness
  - Not a huge number (not hard to fill before you starve)
  - Not a huge number (not unfair to others—everybody does 8)
- “Clean” buffers, fill, queue to disk on your own time

Summary

90%	= 67.5	21 students
80%	= 60.0	7 students
70%	= 52.5	17 students
60%	= 45.0	9 students
<60%		9 students

Comparison

- Usually top two would be flipped, roughly
  - “I get it”, and also some grader gentleness
- Bottom three are essentially last fall's #'s

Implications

Score below 52?

- Figure out what happened
- Probably plan to do better on the final exam

Score below 40?

- Something went *very* wrong
- Passing the final exam may be a serious challenge
- To pass the class you must demonstrate some proficiency on exams (project grades alone are not sufficient)