15-410

"My other car is a cdr" -- Unknown

Exam #1
Oct. 20, 2025

Dave Eckhardt

Babu Pillai

Synchronization

Checkpoint schedule (NOTE NEW HASH FUNCTION)

- Friday during class time
- Meet in Wean 5207
  - If your group number *ends* with
    - 0-2 try to arrive 10:55-11:00 (5 minutes early)
    - 3-5 arrive at 11:13:17
    - 6-9 arrive at 11:31:19
- Preparation
  - Your kernel should be in mygroup/p3ck2
  - We are expecting everybody (even if not quite done)
    - Unless you notify us by noon on Thursday

Synchronization

Checkpoint 2 - alerts

- Reminder: context switch ≠ timer interrupt!
  - Timer interrupt is a *special case*
  - Some timer interrupts will *not* cause context switch
    - Really!
  - Most context-switch invocations will have nothing to do with the timer
    - Really!

Synchronization

Checkpoint 2 - alerts

- Reminder: context switch ≠ timer interrupt!
  - Timer interrupt is a *special case*
  - Some timer interrupts will *not* cause context switch
    - Really!
  - Most context-switch invocations will have nothing to do with the timer
    - Really!
- Please read the handout warnings about context switch and mode switch and IRET *very carefully*
  - Each warning is there because of a big mistake which was very painful for previous students

Synchronization

Book report!

- This your approximately-mid-semester reminder about the book report assignment

Synchronization

Asking for trouble?

- If you aren't using source control, that is probably a mistake
- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - GitHub sometimes goes down!
    - S'13: on P4 hand-in day (really!)
  - Roughly 30% of groups have blank REPOSITORY directories...

Synchronization

Asking for trouble?

- If you aren't using source control, that is probably a mistake
- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - GitHub sometimes goes down!
    - » S'13: on P4 hand-in day (really!)
  - Roughly 30% of groups have blank REPOSITORY directories...
- If your code isn't built and tested on Andrew Linux every two or three days, you are asking for trouble
  - Don't forget about CC=clang / CC=clangalyzer
  - Using a variety of compilers is likely to expose issues

Synchronization

Asking for trouble?

- If you aren't using source control, that is probably a mistake
- If your code isn't in your 410 AFS space every day, you are asking for trouble
  - GitHub sometimes goes down!
    - S'13: on P4 hand-in day (really!)
  - Roughly 30% of groups have blank REPOSITORY directories...
- If your code isn't built and tested on Andrew Linux every two or three days, you are asking for trouble
  - Don't forget about CC=clang / CC=clangalyzer
  - Using a variety of compilers is likely to expose issues
- Running your code on the crash box may be useful
  - But if you aren't doing it fairly regularly, the first "release" may take a *long* time

Synchronization

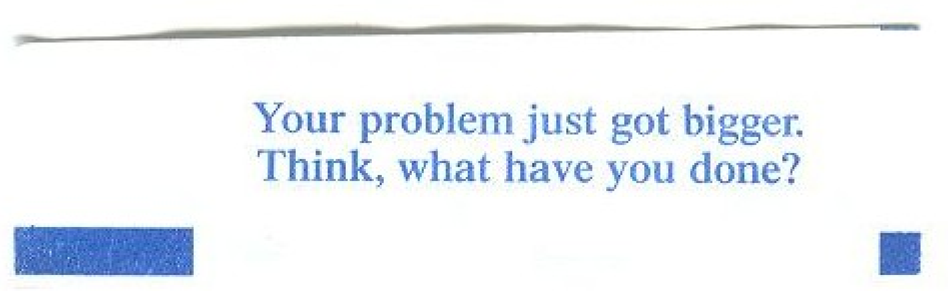
Debugging advice

- Once as I was buying lunch I received a fortune

Synchronization

Debugging advice

- Once as I was buying lunch I received a fortune



Your problem just got bigger.
Think, what have you done?

Image credit: Kartik Subramanian

A Word on the Final Exam

Disclaimer

- Past performance is not a guarantee of future results

The class will change

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

Examination will change to match

- More design questions
- Some things you won't have implemented (text useful!!)
- Still 3 hours, but could be more stuff (~85 points, ~6 questions)

Thanks for Avoiding Faint Pencil!

It wasn't a problem on the mid-term

- Let's keep it that way for the final exam!

“See Course Staff”

If your exam says “see course staff”...

- ...you should!

This generally indicates a serious misconception...

- ...which we fear will seriously harm code you are writing now...
- ...which we believe requires personal counseling, not just a brief note, to clear up.

...though it might instead indicate a complex subtlety...

- ...which we believe will benefit from personal counseling, not just a brief note, to clear up.

“See Instructor”...

- ...means it is probably a good idea to see an instructor...
- ...it does not necessarily imply disaster.

“Low Exam-Score Syndrome”

What if my score is really low????

- It is frequently possible to do *dramatically* better on the final exam
- Specific suggestions later
    - Please execute those instructions in order

Outline

Question 1

Question 2

Question 3

Question 4

Question 5

Q1 – Short Answer

Three parts

- Condition-variable rules
- Top/bottom halves
- Trap vs. interrupt

Q1a – Condition-variable rules

Basic idea: awareness of how cvars should function

- How should they behave?
- How should applications use them?

Information sources

- Synchronization lectures
- Exam-review material

At a very high level

- Threads that should not run should block
- Threads that are unblocked should be able to run
- The three rules given in lecture are less overly abstract

Q1a – Condition-variable rules

Common issue

- Many students discussed cvar internals
- But application code has the responsibility to use cvars responsibly
- Lots of partial credit

Q1b – Top/Bottom Halves

Question goal

- Recall a key principle for dividing code in device drivers

Information sources

- “Hardware” lecture
- Project 1 lecture

Results

- Some very “creative” answers
- Many correct responses

Q1c – Trap vs. Interrupt

Information sources

- “Hardware” lecture

Results

- Responses were generally good
- Try not to mix the two up!

Q1 – Results

Scores

- ~50% of the class scored 7/10 or above (good)
- ~20% of the class scored *below* 5/10

Q2 – Pausable Semaphores

What we were testing

- Primarily: ability to find *and show* race conditions
- Also: knowledge of what a c.s. algorithm should do

Q2 – Pausable Semaphores

What we were testing

- Primarily: ability to find *and show* race conditions
- Also: knowledge of what a c.s. algorithm should do

Cautions

- It is not ok to assume illegal use of a synch object, then show a "race condition"!
- It is good to inspect "`if` vs. `while`", but "every `if` is a bug" is not a rule
- "Every other thread can go before me once" is *the opposite* of a bounded-waiting failure ("once" is a bound!)
- "The pause operation must instantly freeze all other threads" is too strong – showing it doesn't happen isn't showing a failure

Q2 – Pausable Semaphores

Guidance

- One synchronization failure assumes dubious usage by one of the threads
  - This dubious usage does not appear in the sample program
- One synchronization failure is much more likely to occur than the other
- If you found one, finding the other one might be good practice (though the other one might be subtle)

Q2 – Results

Scores

- 54% of the class got 14/15 or 15/15 (good!)
- ~20% of the class scored *below* 10/15 (10/15 == 60%)

Q3 – “Super Semaphores”

Question goal

- Slight modification of typical “write a synchronization object” exam question

Interesting question features

- Can be done *well* with or without aux structs
  - If you solved it one way, maybe try again a different way?
- Some short solutions and some long solutions are reasonable

Q3 – "Super Semaphores"

Question goal

- Slight modification of typical "write a synchronization object" exam question

Interesting question features

- Can be done *well* with or without aux structs
    - If you solved it one way, maybe try again a different way?
- Some short solutions and some long solutions are reasonable
    - Some short solutions are not stellar, though
        - » Piling a bunch of threads up on a mutex for an indefinite period of time is short *but probably turns the fans on*
        - » An rwlock is arguably *anti-good* at stopping threads promptly

Q3 – "Super Semaphores"

Things to watch out for

- Many solutions included starvation (perhaps of threads requesting "too many" thingies)
- There were some progress failures (threads waiting indefinitely despite sufficient thingies being present)
- Does the right thing happen if a `signal()` operation deposits quite a few thingies?
- Avoid "thundering herd" aka "churn"
  - One giant cvar
  - Unbounded number of threads of all types waiting on it for different things
  - `cond_broadcast()` wakes everybody up and many threads must block again
- *When possible*, `cond_signal()/cond_broadcast()` outside of a mutex is better than inside

Q3 – “Super Semaphores”

General note on blocking

- Threads that can't do productive work should *stop running*
- Once stopped, a thread should remain stopped *until there is a reasonable likelihood that it can do productive work*

Q3 – “Super Semaphores”

General note on blocking

- Threads that can't do productive work should *stop running*
- Once stopped, a thread should remain stopped *until there is a reasonable likelihood that it can do productive work*

General conceptual problems

- “x() takes a pointer” does *not* mean “x() must call malloc()”
- Assigning to a function parameter changes the *local copy*
  - It has no effect on the calling function's value
  - C isn't C++ or Pascal (luckily!)
- See course staff about any general conceptual problems revealed by this specific exam question

Q3 – “Super Semaphores”

Approach guidance

- This question mixes counting with blocking for two very-different reasons (but maybe it's *three* different reasons?)
    - Existing primitives implement counting and blocking and unblocking
        - So it is possible to offload lots of work
        - But it is important to keep track of who should receive priority to take various steps
- Pseudo-code/outline *strongly* suggested
    - Pseudo-code/outline all parts before coding any part
    - Consider writing helper functions!
- “First I'll code up wait(), then I'll code up signal()” is much less likely to result in correct code

Q3 – “Super Semaphores”

Important general advice!

- It's a good idea to trace through your code and make sure that at least the simplest cases work without races or threads getting stuck
  - If the question provides example traces, it's prudent to check that your code does the right thing for those traces!

Other things to watch out for

- Memory leaks
- Memory allocation / pointer mistakes
- Forgetting to shut down underlying primitives
- Parallel arrays (use structs instead)

Q3 – “Super Semaphores”

Outcomes

- ~60% of the class scored 16/20 or better (80%+)
  - This question is arguably “not super hard”
- ~20% of the class “did not do ok” (under 60%)
  - These outcomes are concerning

Other questions in this category are harder

- Perhaps a final-exam question might be harder

Q4 – Blocking (“Process Model”)

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is *not in a run queue*

Q4 – Blocking (“Process Model”)

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is ***not in a run queue***

Dangerous idea

- “If a thread invokes `gettid()`, the thread's execution is suspended until the system call returns.”
  - This is dangerously wrong.
  - The thread isn't suspended: it's ***running*** gettid()!

Q4 – Blocking ("Process Model")

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is *not in a run queue*

Dangerous idea

- "If a thread invokes `gettid()`, the thread's execution is suspended until the system call returns."
  - This is dangerously wrong.
  - The thread isn't suspended: it's *running* gettid()!

Common misconception

- Question text reminds: especially on a multiprocessor, "might need a lock" does *not* mean "likely to block"
  - Remember that we assume most locks are *usually not contested* and are *held briefly*

Q4 – Blocking (“Process Model”)

For full credit

- Blocked thread can't run until a specific event
- Blocked thread is *not in a run queue*

Dangerous idea

- “If a thread invokes `gettid()`, the thread's execution is suspended until the system call returns.”
  - This is dangerously wrong.
  - The thread isn't suspended: it's *running* gettid()!

Common misconception

- Question text reminds: especially on a multiprocessor, “might need a lock” does *not* mean “likely to block”
  - Remember that we assume most locks are *usually not contested* and are *held briefly*
  - *Sometimes* we use a synch object that blocks threads, but locking and blocking are not the same thing

Q4 – Blocking (“Process Model”)

Common glitches

- Vagueness about non-runnability (common deduction: “-1 OOQ”)
- Explaining why *part* of `new_pages()` should be straightforward
    - There are two other parts!

Q4 – Blocking (“Process Model”)

Common glitches

- Vagueness about non-runnability (common deduction: “-1 OOQ”)
- Explaining why *part* of `new_pages()` should be straightforward
  - There are two other parts!

The “hierarchy”

- Running and doing useful work (user mode *or kernel mode*)
- [Running and doing “locking work”]
- Runnable but not running (in scheduler “run queue”)
- Blocked = not running and not runnable

Q4 – Blocking (“Process Model”)

Results

- Many students got 8/10 or better
- Scores below 7/10 are concerning
  - Blocking is a key concept

Q5a – Nuts & Bolts: “capture %eip”

Purpose: Think about using familiar asm instructions in unfamiliar ways.

- Can be solved with one or two lines of code
- Two approaches
  - Use a (very) common instruction that manipuates %eip
  - Use linker's ability to assign absolute addresses to symbols

Outcomes

- Reasonable distribution of scores
- Not legal to use %eip as an instruction argument (x86-32)
- Partial credit given for some kind of valid %eip manipulation

Q5b – Nuts & Bolts: variable locations

Purpose: Review your understanding of a basic idea.

- 2 in BSS
- 1 in data
- 3 in stack (2 in a special place)

Outcomes

- This should be an easy/fast question
  - For the rest of the semester you will spend a lot of time debugging stacks!
- Some perfect scores, but arguably not enough

Q5 – Results

Overall outcomes

- ~30% got 10/10
- Scores under 8/10 (1/6 of class) are arguably concerning

Breakdown

90% = 58.5	5 students	(57.0 and up)
80% = 52.0	10 students	(52.0 to 56.0)
70% = 45.5	7 students	(44.0 to 51.0)
60% = 39.0	3 students	(38.0 to 43.0)
50% = 32.5	1 student	(31.0 to 37.0)
40% = 26.0	0 students	
<40%	1 student	

Comparison/calibration

- Overall scores don't look blatantly problematic

Implications

Score below 50?

- Form a “theory of what happened”
  - Not enough textbook time?
  - Not enough reading of partner's code?
  - Lecture examples “read” but not grasped?
  - Sample exams “scanned” but not solved?
- It is important to do better on the final exam

Implications

Score below 50?

- Form a “theory of what happened”
  - Not enough textbook time?
  - Not enough reading of partner's code?
  - Lecture examples “read” but not grasped?
  - Sample exams “scanned” but not solved?
- It is important to do better on the final exam
  - Historically, an explicit plan works *much* better than “I'll try harder”
  - *Strong suggestion*:
    - » Identify causes, draft a plan, see instructor

Implications

Score below 40?

- Something went *noticeably* wrong
  - It's *important* to figure out what!
- Beware of “triple whammy”
  - Low score on three “core” questions
  - Generally Q2, Q3, Q4
- Passing the final exam could be a challenge
- *Passing the class may be at risk!*
  - To pass the class you must demonstrate proficiency on exams (not just project grades)

Implications

Score below 40?

- Something went *noticeably* wrong
    - It's *important* to figure out what!
- Beware of “triple whammy”
    - Low score on three “core” questions
    - Generally Q2, Q3, Q4
- Passing the final exam could be a challenge
- *Passing the class may be at risk!*
    - To pass the class you must demonstrate proficiency on exams (not just project grades)
- Try to identify causes, draft a plan, see instructor
    - Good news: explicit, actionable plans usually work well

Action plan

Please follow steps in order:

1. Identify causes
2. Draft a plan
3. See instructor

Action plan

Please follow steps in order:

1. Identify causes
2. Draft a plan
3. See instructor

Please avoid:

- "I am worried about my exam, what should I do?"
  - *Each person should do something different!*
  - The "identify causes" and "draft a plan" steps are individual, and depend on some things not known by us

Action plan

Please follow steps in order:

1. Identify causes
2. Draft a plan
3. See instructor

Please avoid:

- “I am worried about my exam, what should I do?”
  - *Each person should do something different!*
  - The “identify causes” and “draft a plan” steps are individual, and depend on some things not known by us

General plea

- Please check to see whether there is something we strongly recommend that you have been skipping because you never needed to do that thing before
  - This class is different