15-410
“...misbehave(7)...”

Project 2
Sep. 19, 2018

Dave Eckhardt
Dave O'Hallaron
Synchronization

Project 2 out today

- Writeup this afternoon
  - Please read carefully!
- Tarball afternoon/evening
  - Feeling impatient? Consider reading the writeup again
- Group volumes should be ready this afternoon

Please make sure you've discussed with your partner

- How many late days?
- Project *schedule* in other classes
  - Write down a joint project schedule
- Auditing or pass/fail? Target 410 grade?
- Prior experience
- Interviews
Outline

What you'll build
- Mutex, condition variable
- Thread library
- Supplemental library routines
- Tests

How the pieces fit together
- A picture is worth 1000 words
- You'll need to read the handouts too
  - (two, each >1000 words)
  - kspec – specifies our kernel for P2, your kernel for P3
  - thr_lib – specifies thread library
Mutex & Condition Variable

mutex    cvar

410 kernel
Remainder of Thread Library

thr_create()  thr_exit()  ...

mutex  cvar

410 kernel
Supplemental Library Routines

- r/w lock
- semaphore
- thr_create()
- thr_exit()
- ...
- mutex
- cvar
- 410 kernel
Tests (Yours & Ours)

- user tests
- 410 tests
- r/w lock
- semaphore
- thr_create()
- thr_exit()
- ...
- mutex
- cvar
- 410 kernel
Building a “RAM disk” image

- test1.o → libthread → libstdio → test1
- 410test1.o → libthread → libstdio → 410test1
- user_apps.o

Diagram showing the flow of dependencies and linking for building a RAM disk image.
Linking “RAM disk” to kernel

user_apps.o

kernel.o

boot image
Misbehave

misbehave(int mode)

- Special debugging-support system call in our 410 kernel
- Adjusts “behavior” of system
  - Multiple legal behaviors (you will feel this during P3)
  - Each mode selects a particular mix
  - We will not document these
  - We expect you to not “document” them to classmates either
- Debug your thread library with one mode, then the next...
  - A dazzling array of flavors
  - 0...63
  - maybe even more
  - -1
- You will not be required to implement misbehave() in P3
threadinfo

simics> tidinfo 11
REGISTER DUMP FOLLOWS
   CS = 0x00000043, EFLAGS = 0x00010246, SS = 0x0000004b
   EIP = 0x0100004a, ESP = 0xfffffffffa0, EBP = 0xffffffffcc
   EDI = 0x00000000, ESI = 0x00000000, EAX = 0x31337000
   EBX = 0x00000000, ECX = 0x00000000, EDX = 0x01000c0a

Cool, what is it?
- Debugging information about thread 11
- The last instruction it executed in user space

Why would I want that?
- It might help with certain hard problems
Plea – Conceptual

This code is *tricky*

- Most of you have already written multi-threaded code
  - That can be tricky enough
- Writing the internals is harder
  - Get a part 99% done
  - Discover a “bug”...
  - ...which is really a misconception...
  - *Totally new design* to fix it

Make sure core parts are *solid*

- Better to skip readers/writers locks if not
Plea – Time

The first 90% will take the first 90% of the time
  • The last 10% will take the second 90% of the time

“Code complete”
  • Plan to spend at least three days debugging based on the tests we release
  • If your thread library doesn't pass cyclone and agility_drill it won't pass a bunch of our tests either
    • Resultant grade is unlikely to exceed a C

“You should be here” guidance in handout
  • Based on bitter experiences of former students