Welcome ~~back~~ to CS439!

Introductions

Neil, Alex H, Jerry, and Alex M

Why care about this class

- Literally everything runs on an operating system
- Understanding how your underlying infrastructure works allows you to write better and more efficient code
- Even if you never touch this stuff again you'll be able to understand more nuances of how your code works
- It's reaaaaaaaaaally fun :3

Logistical stuff

Logistical stuff

- FERPA (...and why you should care)
  - Student confidentiality
  - Your information is confidential and will not be shared
  - You cannot share class material publicly
- Title IX (...and why you should care)
  - ALL teaching assistants and professors are mandatory reporters for Title IX
  - If we hear anything, we are required to report incidents of sex discrimination, sexual harassment, sexual assault, sexual misconduct, interpersonal violence, and stalking

Stress

- 439H is **not an easy class**
  - Lots of new material
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer or solution
  - This is expected - we want everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overwhelmed or spending more time on this class than you think you should be, **please reach out** to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources if we are not able to help

Mental health resources available at UT

Logistical stuff

- Ed Discussion
  - Piazza is gone :(
  - We will be posting announcements and clarifications here
  - Notifications are iffy, please check the site regularly!
  - Please use private posts for anything specific to your code or your grades
  - Otherwise please post publicly so others can benefit from your question!
  - Help answer others' questions!
  - Private posts are also a great way to contact us + Dr. Gheith

Logistical stuff

- Office hours
  - TBD, but should be finalized by next week
  - We will post a poll on Ed discussion about possible times

Logistical stuff

- Grading
  - Assignments
    - (Mostly) one every week, rubric may differ slightly per week
    - P1 test case due Sunday, submission due Tuesday
  - Quizzes
    - One every **two** weeks
    - Less quizzes (yay)! Each quiz is weighed more (boo)!
  - Participation
    - Scribe notes - like last semester, post **4** sets of notes throughout the semester
    - Please show up to class and show up to office hours! It will help a lot!

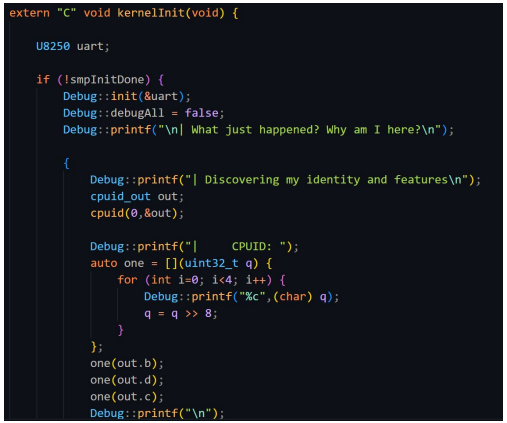
Logistical stuff

- Regrade requests
  - Make a private note on Ed discussion
  - Tag it with Regrades - we may not get around to it otherwise
  - Include your name, eid, which assignment, any other info we might need to figure out what you actually want graded
  - Include a description of why you think something should be regraded (justify yourself)
  - Please do not submit regrade requests just because you’re hoping to get more points

P1

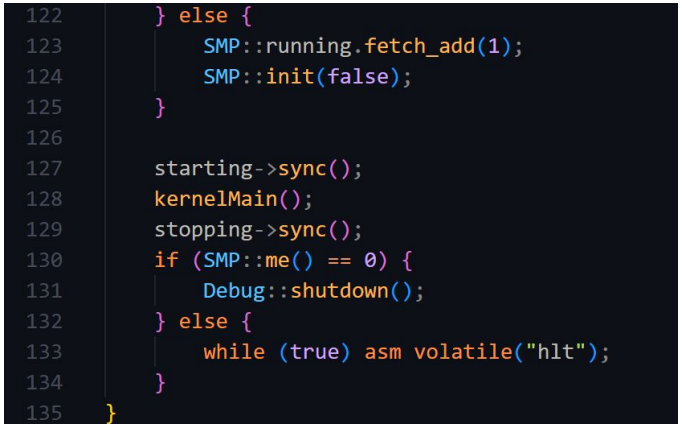
P1

- Understanding the kernel code:
  - There's a lot to take in... but try to understand the general purpose of each file
  - We will be building on this code throughout the semester
  - `init.cc` is a good place to start
  - `atomic.h` has some primitives that might be useful for this week



```
extern "C" void kernelInit(void) {

    U8250 uart;

    if (!smpInitDone) {
        Debug::init(&uart);
        Debug::debugAll = false;
        Debug::printf("\n| What just happened? Why am I here?\n");

        {
            Debug::printf("| Discovering my identity and features\n");
            cpuid_out out;
            cpuid(0,&out);

            Debug::printf("|     CPUID: ");
            auto one = [](uint32_t q) {
                for (int i=0; i<4; i++) {
                    Debug::printf("%c",(char) q);
                    q = q >> 8;
                }
            };
            one(out.b);
            one(out.d);
            one(out.c);
            Debug::printf("\n");
```

Kernel initialization stuff



```
122         } else {
123             SMP::running.fetch_add(1);
124             SMP::init(false);
125         }
126
127         starting->sync();
128         kernelMain();
129         stopping->sync();
130         if (SMP::me() == 0) {
131             Debug::shutdown();
132         } else {
133             while (true) asm volatile("hlt");
134         }
135     }
```

Call to kernelMain (implemented by test case)

P1

- What will the final value of x be?

```
int x = 0;

/* Called by 2 cores */

void kernelMain(void) {

    x++;

}
```

P1

- What will the final value of x be?

```
int x = 0;
/* Called by 2 cores */
void kernelMain(void) {
    x++;
}
```

Core 0	Core 1
`load x`	`load x`
`add x, 1`	`add x, 1`
`store x`	`store x`

P1

- What are atomics?
  - Allow for “instantaneous”* operations
  - Entire operation is guaranteed to happen without anything in between
  - Only good for small operations
  - Useful to build synchronization primitives
  - Usually implemented at the hardware level

```
template <typename T>
class Atomic {
    volatile T value;
public:
    Atomic(T x) : value(x) {}
    Atomic<T>& operator= (T v) {
        __atomic_store_n(&value,v,__ATOMIC_SEQ_CST);
        return *this;
    }
    operator T () const {
        return __atomic_load_n(&value,__ATOMIC_SEQ_CST);
    }
    T fetch_add(T inc) {
        return __atomic_fetch_add(&value,inc,__ATOMIC_SEQ_CST);
    }
    T add_fetch(T inc) {
        return __atomic_add_fetch(&value,inc,__ATOMIC_SEQ_CST);
    }
    void set(T inc) {
        return __atomic_store_n(&value,inc,__ATOMIC_SEQ_CST);
    }
    T get(void) {
        return __atomic_load_n(&value,__ATOMIC_SEQ_CST);
    }
    T exchange(T v) {
        T ret;
        __atomic_exchange(&value,&v,&ret,__ATOMIC_SEQ_CST);
        return ret;
    }
    void monitor_value() {
        monitor((uintptr_t)&value);
    }
};
```

*The operations aren’t actually instantaneous. There are a few different models on what this means but for our class we can safely assume that an atomic operation will finish in its entirety before any other operation modifies any part of memory which is touched by anything in the atomic operation.

P1

- What are locks?
  - Many ways to implement
  - This week, we have a SpinLock implemented for us
  - Uses atomic operations to allow one core in at a time
  - The other cores “spin” on a while loop

```
class SpinLock {
    Atomic<bool> taken;
public:
    SpinLock() : taken(false) {}

    SpinLock(const SpinLock&) = delete;

    // for debugging, etc. Allows false positives
    bool isMine() {
        return taken.get();
    }

    void lock(void) {
        taken.monitor_value();
        while (taken.exchange(true)) {
            iAmStuckInALoop(true);
            taken.monitor_value();
        }
    }

    void unlock(void) {
        taken.set(false);
    }
};
```

P1

- Critical sections
  - Want to guarantee that only **one** core is in the critical section at a time
    - Why is this useful?
  - How to do this? Synchronization primitives
  - Many different possibilities, but utilize the limited resources you're given
    - No c++ library
    - But you have the kernel functions
    - And you can always implement your own features
  - How do you distinguish the identity of a core?
  - Perhaps execute different code based on core identity?

P1

- `critical.h`
  - Implement the `critical` function, do `work` inside a critical section
- `critical.cc`
  - Has some global variables to help you with your implementation
  - These are optional; you can use none or all of them or even add more
- These are the only two files you should be modifying for this project
- Don’t modify the outward-facing API, the test cases will all call the function in the same way

P1

- Test case quality
  - In general, when writing test cases, try to test something interesting
  - Tests should either extensively test a small portion of the spec, or test multiple aspects of the spec
  - Tests should be designed to help the implementer improve their code
    - Add helpful comments to describe what your test is doing
    - Make your tests modular
    - Have clear and concise printouts
  - This may be hard for P1 since there isn't that much to test, but try your best

Questions?