Multithreaded code – mutexes, atomics, when things go wrong

```
Critical sections
                                              - if var is read only everywhere
locking?
                                                then you don't need locking
     mutex 0 if concerned with
                                              - Us soon as you have a single write
     atomics 1) speed
                                                (in a multithreaded situation) you
                                                 need locking
 Mutex
                                              - Use atomics when crit section
     -risks of exceptions skipping
                                                 is a single line of code (like a
                                                 global counter). If more than I line
     - RAIT
                                                  or a complex object ( like a filestream)
                                                  then use a mutex
```

demo RAII for mutex (create a class that takes a pointer to a mutex and locks in constructor and unlocks in destructor) point out flaws, (exceptions, dont lock in constructor, unlock in destructor, what if shallow copied pointer from constructor is not there when destructor calls it)

Tell them to use lock_guard<mutex> m(amutex); does all RAII class does in an exception safe way

talk a little about templates in atomics (take any type) show page where standard types are http://www.cplusplus.com/reference/atomic/atomic/?kw=atomic atomics are lighter weight than mutexes, faster to lock and less code to write

When things go wrong

lock a mutex twice join a thread twice deadlock (2 mutexes)

how to tell where a locked program is? In IDE, set breakpoint, debug until you hit it, then resume. When it locks hit pause button to see where threads are.

Lots more- see in Operating Systems