Lecture 9: Concurrency Conclusion

9.0 Main Points:

Summarize concurrency section
Illustrate drawbacks to threads

9.1 Concurrency Summary

Basic idea in all of computer science is to abstract complexity behind clean interfaces. We've done that!

Physical Hardware	Programming Abstraction
single CPU, interrupts, test&set	sequential execution, infinite # of CPUs, semaphores and monitors

Every major operating system built since 1985 has provided threads -- Mach, OS/2, NT (Microsoft), Solaris (new OS from SUN), OSF (DEC Alphas). Why? Because makes it a lot easier to write concurrent programs, from Web servers, to databases, to embedded systems.

So does this mean you should all go out and use threads?

9.2 Cautionary Tales

Illustrate why an abstraction doesn't always work the way you want it to.

9.2.1 OS/2

Microsoft OS/2 (around 1988): initially, a spectacular failure. Since then, IBM has completely re-written it from scratch.

Used threads for everything -- window systems, communication between programs, etc. Threads are a good idea, right?

Thus, system created lots of threads, but few actually running at any one time -- most waiting around for user to type in a window, or for a network packet to arrive.

Might have 90 threads, but just a few at any one time on the ready queue. And each thread needs its own execution stack, say, 9KB, whether it is runnable or waiting.

Result: system needs an extra **1 MB** of memory, mostly consumed by waiting threads. 1 MB of memory cost $200 in 1988.

Put yourself in the customer's shoes. Did OS/2 run Excel or Word better? OK, it gave you the ability to keep working when you use the printer, but is that worth $200?

Moral: threads are cheap, but they're not free.

Who are operating systems features for?
Operating system developer?
End user?

Lots of operating systems research has been focused on making it easier for operating systems **developers**, because it is so complicated to build operating systems.

But the trick to selling it is to make it better for the **end user**.

9.2.2 Threads and Multiprocessors

Might think you have everything you need to know to go write a parallel program: Just split program up into threads, so that things can run in parallel.

Example: Matrix multiply

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
     for (k = 0; k < N; k++)
        C[i][j] += A[i][k] * B[j][k];
```

How would you parallelize this? Create a thread for every iteration of inner loop? Each one can run concurrently, using a lock to protect access to each element in C[i][j].

Would work, but wouldn't be efficient. In Nachos, a few hundred instructions to create a thread. Here, maybe ten instructions to do each iteration.

Repeat: threads are cheap, but they aren't free.

Instead: group iterations so that each thread does a fair amount of work.