Lecture #5: Independent and cooperating threads

****
Review -- 1 min
****

multi-threaded process

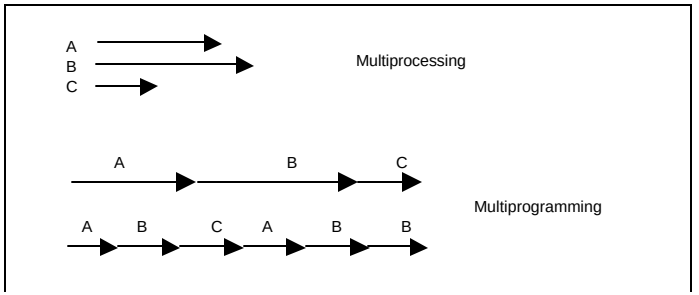
User-level v. kernel threads
Pre-emptive v. non-pre-emptive threads
Thread control block
Dispatch

****
Outline - 1 min
****
Finish discussion of thread creation + dispatch
Independent v. cooperating threads
Atomic operations

****
Preview - 1 min
****
Abstraction dilemma – want "independence" and "cooperation"

****
Lecture - 20 min
****

1. Multiprocessing v. Multiprogramming



Dispatcher can choose to run each thread to completion
or
time-slice in big chunks
or
time-slice so that each thread executes only one instruction at a time (simulating a multiprocessor where each CPU operates in lockstep)
If the dispatcher can do any of the above – programs must work under all cases, for all interleavings

So how can you know if your concurrent program works?
Whether **all** interleavings will work?

Option 1: enumerate and test all possibilities
Impossible!
Option 2: maintain *invariants* on program state; structure program carefully to maintain these invariants

****
Admin - 3 min
****

Feedback on project 2
Project 3 available

****
Lecture - 33 min
****

2. Independent v. cooperating threads

2.1 Definitions

Independent threads – no shared state with other threads
- deterministic – input state determines result
- reproducible
- scheduling order doesn't matter

cooperating threads – share state
- non-deterministic
- non-reproducible

Non-reproducibility and non-determinism means that bugs can be intermittent. ***This makes debugging hard***.

2.2 Why allow cooperating threads?

People cooperate; computers model people's behavior, so at some level they have to cooperate

1. Share resources/information
   a) one computer, many users
   b) one bank balance many tellers
2. Speedup
   a) overlap I/O and computation
   b) multiprocessors – chop up program into little pieces and run them in parallel
3. Modularity
   Chop up large problem into simpler pieces

   for example – typesetting: `ref | grn | tbl | eqn | troff`

   This makes the system easier to extend; you can write eqn without changing troff
4. Fundamentally required – look at thread switch example above – different threads share ready queue, scheduling data structures, ...

2.3 Some simple concurrent programs

Most of the time, threads are working on separate data, so scheduling order doesn't matter

Thread A	Thread B
x = 1;	y = 2;

What are the possible values for x:

x = 1;	x = 2;

What are the possible values for x:
initially: y = 12

x= y + 1;	y = y * 2;

What are the possible values for x:
initially x = 0

x = x + 1;	x = x + 2;

2.4 Atomic operations

atomic operation – always runs to completion or not at all; indivisible. Can't be stopped in the middle.

On most machines, memory reference and assignment (load and store) of **words** are atomic

Many instructions are not atomic. For example, on most 32-bit architectures, double precision floating point store is not atomic. It involves 2 separate memory operations.

2.5 A larger concurrent program example

Two threads, A and B, compete with each other. One tries to increment a shared counter, the other tries to decrement the counter.
For this example, assume that memory load and memory store are atomic, but incrementing and decrementing are **not** atomic

```
Thread A                    Thread B
I = 0;                      I = 0;
while(I < 10){              while(I > -10){
   I = I + 1;                       I = I - 1;
}                           }
print "A wins"              print B wins
```

QUESTIONS
1) Who wins?
→ could be either

2) Is it guaranteed that someone wins? Why or why not?

3) What if both threads have their own CPU, running in parallel at exactly the same speed. Is it guaranteed that it goes on forever?

In fact, if they start at the same time, with a ½ an instruction ahead, B will win quickly

4) Could this happen on a uniprocessor?

Yes! Unlikely, but if you depend on it **not** happening, it will eventually happen, and your system will break and it will be very difficult to figure out why.

****
Summary - 1 min
****

Thread programming – nondeterministic, irreproducible, intuition not always a good guide

I repeat: it is **impossible** to enumerate and reason about all possible interleavings!

Key notions

Invariants – facts that must always hold true
atomic actions – the only thing you can trust

Next 2 weeks – learn how to structure program so that we can use atomic actions to build higher level programs that have invariants about which we can reason