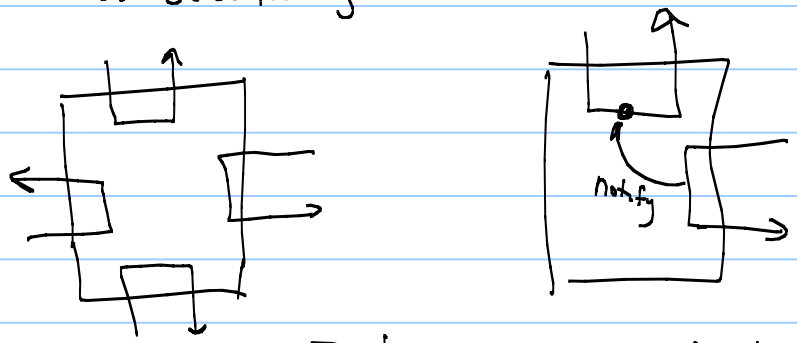
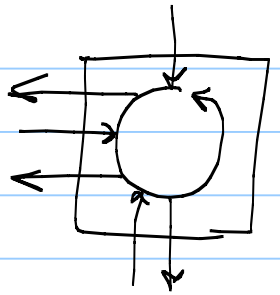
Lecture 12

Note Title 2/19/2008

Bounded overtaking?



Faster - less overhead



one thread touching data:

- inherently sequential
- can handle background bookkeeping
- controls the order that msgs are processed
- no issue about re-entrant locks

Variations on monitors:

- notify and Exit vs notify and Continue

The notify operation includes leaving the monitor. — higher chance that the condition being waited for is still true.

Multiple wait queues – avoid some spurious wakeups – simplifies programmy in that case as well

Concurrency in Java

- To create a Thread (process in OS terms)
  - subclass the Thread class with a run() method that represents the desired behavior.
  - instantiate an object of that class
    new Foo().start()
  - Alternatively any class can implement Runnable interface, include a run() method. and then it can be start() 'ed

in Oz or ML

The data for a thread comes along w/ the function that is passed to spawn.

in Java — The data belongs to the object.

Monitors

synchronized keyword applied to a method means: when method is called acquire the lock associated with every object; release the lock when returning.

or synchronized (object) { block }

Every object also has a wait queue:
in order to wait on it
wait() only while holding the lock.
To get off wait queue another thread must
execute notify() or notifyAll().

while() wait(timeout) ....

Rules for shared variables in Java:

Atomicity: at what level is a store atomic?

field values except for long and double are atomic

also changes made in a synchronized block are atomic wrt any other synchronized block using the same lock.

Visibility of changes:

When do changes by one thread become visible to another?

Again, synchronized blocks do what you expect.

Order: what order do changes appear to occur on a different thread?

synchronized blocks appear to run in order

Beyond these rules – almost no guarantees!

Why does knowing this stuff help you?

Double-checked locking

```
class Foo {
  private Helper helper = null;
  public synchronized getHelper () {
      if (helper == null) {
          helper = new Helper()}
      return helper;
  }
}
```

Double-checked locking

```
class Foo {
    private Helper helper = null;
    public getHelper() {
        if (helper == null) {
            synchronized (this) {
            if (helper == null) {
                helper = new Helper(); }
            return (helper);
        }
    }
}
```

Concurrent programming in C and C++:

what is the storage model?

- depends on compiler
- processor
- run-time support system & OS

```
class Buffer {
int [] buf;
int first, last, size, i;
public      Buffer(int n) {
    buf = new int[n];
     size = n; first = 0; last = 0; i = 0;
}
```

i == 0 empty.
i == n full

invariants:
→ first == last ≡ buffer is empty or full?
first = last+1 ≡ full

first pts to first available
last ptr to first place to put.

```
public synchronized void put(int x){
→  while(i==n) wait();
   buf[last] = x;
   last = (last+1) % size;
   i = i+1;
   notifyALL();
}
```

```
int get() {
→  while(i==0) wait();
   x = buf[first]
   first=(first+1) % size;
   i = i-1;
   notify();
   return x
}
```