Tricks (mixed syntax)

```
if (some_condition) // as a hint
{
  LOCK m DO
     if (some_condition) //the truth
     {stuff}
  END
}
```

Cheap to get info but must check for correctness; always a slow way

More Tricks

General pattern:

```
  while (! required_conditions)  wait (m, c);
```

Broadcast works because waking up too many is OK (correctness-wise) although a performance impact.

```
 LOCK m DO
    …
    deferred_signal = true;
 END
 if (deferred_signal) signal (c);
```

Spurious lock conflicts caused by signals inside critical section and threads waking up to test mutex before it gets released.

Alerts

- Thread state contains flag, alert-pending
- Exception alerted
- Alert (thread)
  - alert-pending to true, wakeup a waiting thread
- AlertWait (mutex, condition)
  - if alert-pending set to false and raise exception
  - else wait as usual
- Boolean b = TestAlert()
  - tests and clear alert-pending

```
TRY
   while (empty)
     AlertWait (m,
   nonempty); return
   (nextchar());
EXCEPT
   Thread.Alerted:
        return (eof);
```

Using Alerts

```
sibling = Fork (proc, arg);
while (!done)
{ done = longComp();
   if (done) Alert (sibling);
   else done = TestAlert();
}
```

Wisdom

Do s

- Reserve using alerts for when you don’t know what is going on
- Only use if you forked the thread
- Impose an ordering on lock acquisition
- Write down invariants that should be true when locks aren’t being held
- Worry about correctness first before performance

Don’t s

- Call into a different abstraction level while holding a lock
- Move the “last” signal beyond scope of Lock
- Acquire lock, fork, and let child release lock
- Expect priority inheritance since few implementations
- Pack data and expect fine grain locking to work

Practice: Producer/Consumer Pipelines

In the fall of 1996, Hurricane Fran hit Durham, which is why you still see so many trees down in the woods. For the sake of local history, here is a Hurricane Fran updating of the traditional cigarette smoker’s problem.

Three men are in a neighborhood with hundreds of fallen trees. To cut up a fallen tree, each man needs all three of the following items: a chainsaw with a sharp chain, gasoline mix, and chain lubricating oil. There is a loggers’ supplier on the web with new chains, gas, and oil in ample quantity making deliveries to the area. Each neighbor has his own chainsaw but only one of them has a sharpener to keep his chain sharp. Another neighbor has a large tank of the gasoline mixture. The third neighbor has lots of lubricating oil.

The action begins when a delivery of two of these items arrives in the neighborhood which would allow one of the neighbors to cut up a tree until the delivered items become exhausted (e.g. for the third neighbor this would mean that his chain gets dull and his gas is gone, but he still has enough oil). When the lucky neighbor who got the benefit of the last shipment is done, he places another order to the supplier, which sends two more items (at random), thus enabling another of the neighbors to obtain all three items and cut up a tree.

Practice: Klingon Problem

The Klingons are attacking. The Federation vessels can escape through the wormhole, but sensors indicate that the wormhole is unstable. The ships' captains plan to create a subspace distortion to prevent the wormhole from closing on them while they are in it. To do this, they will enter the wormhole three at a time while emitting a tachyon pulse through their main deflector dishes. Each ship must lower its shields before initiating its tachyon pulse, and once a ship starts emitting tachyons it can have no contact with the other ships.

Implement a synchronization scheme to allow the Federation to retreat through the wormhole in an orderly fashion. Your scheme should have the property that no ship lowers its shields until just before it enters the wormhole, no ship enters the wormhole until two others are ready to go in with it, and all ships in each group of three enter the wormhole before any of the ships in the next group.

Practice: Fine grain locking

Multiple threads inserting and deleting in a linked list

Practice: Bridge Problem

Synchronize traffic over a narrow light-duty bridge on a public highway. Traffic may only cross the bridge in one direction at a time, and if there are ever more than 3 vehicles on the bridge at one time, it will collapse under their weight. Each car is to be represented by one thread, which executes the procedure OneVehicle in order to cross the bridge:

```
OneVehicle(int direc) //direc is either 0 or 1;
                      //giving the direction in which the car is to cross
{
ArriveBridge(direc);
CrossBridge(direc);
ExitBridge(direc);
}
```