Announcements

- reading
  - for Tuesday 5.5
- Homework #1 (due 9/30/97 in class)
  - ch1, p.4 simple expression and explanation is fine
  - ch 2, p14: just use dvision (assume mean is extact)
- Programming Project #1 was returned

copyright 1997 Jeffrey K. Hollingsworth

Using Locks for the Critical Section

- Lock:
  - if no thread has the lock mark it locked and return
  - if another thread has the lock, wait
- Unlock:
  - release the lock
  - if other threads waiting, notify one **or** all of them
- Called mutexs in pthreads
  - pthread_mutex is the data type
  - pthread_mutex_init used to initialize it
  - pthread_mutex_lock locks it
  - pthread_mutex_unlock releases it
- Lock Grainularity
  - want to lock enough to protect accesses
  - don't want to lock too much to slow down the program

copyright 1997 Jeffrey K. Hollingsworth

Condition Variables

- Allow threads to wait on the value of a variable
  - wait until the list is non-empty for example
  - allows one thread to signal to another thread that something has changed
    - threads may sleep waiting to be notified of this change
- Can unlock and re-lock a mutex before/after suspend

```
wait for count to be >= 1
   pthread_mutex_lock(&count_mutex);
   while (count <= 0) {
      pthread_cond_wait(&count_condvar, &count_mutex);
   }
   pthread_unlcok(&count_mutex);

update count:
   pthread_mutex_lock(&count_mutex);
   count++;
   pthread_mutex_unlock(&count_mutex);
   pthread_cond_signal(&count_condvar);
```


copyright 1997 Jeffrey K. Hollingsworth

Queue Project

- Need to coordinate access to shared resources
  - use mutex to guard access to a shared data structure
- Queue abstraction is **very** useful
  - enqueue: add item to queue
  - dequeue: remove item, **block** if not ready
  - head: return head of queue without dequeue
  - probe: test if the queue is empty
  - must use a mutex to protect access to queue
  - build a producer/consumer test program
- Multiple application threads
  - our test application is multi-threaded
  - must be able to support multiple threads trying to enque

copyright 1997 Jeffrey K. Hollingsworth

Network Layer

- Responsibility
  - end-to-end delivery of packets to the network
  - selecting routes for the packets to take
    - implies knowledge of the network topology
  - managing utilization of the links
    - provide flow control (across multiple links)
    - spread load among different routes
- Interface Design
  - should be independent of subnet technology
  - hide number, type, and topology of network from upper layers
  - export a common number plan for entire network

copyright 1997 Jeffrey K. Hollingsworth

Connection vs. Connectionless

- Two possible designs for network layer
  - connection oriented service (ATM)
    - based on experience of telcos
  - connectionless service (IP)
    - based on packet switching (ARPANET)
- Connectionless
  - transport datagrams from source to destination
    - end-point addresses in every datagram
  - less complex network layer, more complex transport
- Connection oriented
  - also called virtual circuits
  - establish an end-to-end connection with network state
    - can use VCI (global or next hop) in each packet

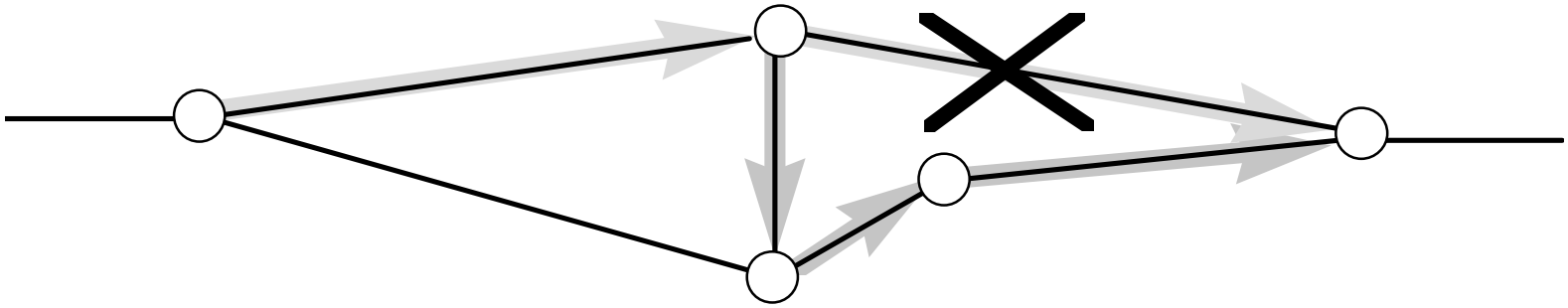
copyright 1997 Jeffrey K. Hollingsworth

Datagram vs. VC Addresses

- Datagrams
  - must include full address in each packet
  - addresses must be unqiue for entire network
    - don't re-use too often
    - addresses per src/dest pair
- Virtual Circuit
  - globally unique
    - requires allocation scheme to ensure its unique
    - consumes many bits per packet
  - per link
    - requires translation at each switch
    - uses fewer bits (important for small packets like ATM)

copyright 1997 Jeffrey K. Hollingsworth

Link Failue in Virtual Circuits



- Re-establish virtual circuit
  - router near failure can patch up link
  - original host/router creates new virtual circuit
- Virtual circuit is dropped
  - transport layer can handle recovery

copyright 1997 Jeffrey K. Hollingsworth

Virtual Circuit vs. Datagram

Issue	Datagram	Virtual Circuit
Circuit setup	not needed	necessary
Addresses	full source/dest per packet	next hop vc sufficient
state	no state in network	per connection data at each router
routing	each packet individually	once at VC setup
router/link failure	a few packets may be lost	all VCs through router are terminated
congestion control	difficult	many pre-allocation and policing policies permitted

copyright 1997 Jeffrey K. Hollingsworth

Routing: Goals

- Correctness
  - packets get where they are supposed
- Simplicity
  - easy to implement correctly
  - possible to make routing choices fast (or updates easy)
- Robustness
  - failures in the network still permit communication
- Stability
  - small changes in link availability results in a small change in the routing information
- Fairness
  - each host, VC, or datagram has the same chance
- Optimality
  - best possible route
  - best utilization of bandwidth

copyright 1997 Jeffrey K. Hollingsworth