Non-multiprocessing approaches to responsiveness

11/2/15
Chat program due Wednesday
Recall: Responsiveness problem

• Our sample echo server is an *iterative server*, completely handling one client before accepting another
  – Will be non-responsive if one request takes a “long” time

• At some level, non-responsiveness is an issue caused by calls that block (server can’t help clients if its blocked)
Last time...

• Creating new process for each connection
• Creating new thread for each connection
• Creating bounded number of processes (or threads) and having them all block on the server socket waiting for connections
An alternative: select

• Instead of waiting on just one socket, wait on a set of them. Unblock as soon as any socket in the set is ready

• Once select tells you that a socket is ready, make a blocking call which is guaranteed not to block (that time)
Demo: TCP-EchoServer-Select.c

Uses select to wait for connections on multiple ports
Can TCPEchoServer-Select respond to multiple simultaneous echo clients at the same time? (i.e. does it resolve the problem we talked about Friday?)

A. Yes
B. Yes as long as there are \( \leq 5 \) clients at a time
C. No because it still blocks on `recv`
D. No because it now blocks somewhere else
E. What did we talk about on Friday again?
Can TCPEchoServer-Select respond to multiple simultaneous echo clients at the same time? (i.e. does it resolve the problem we talked about Friday?)

A. Yes
B. Yes as long as there are ≤ 5 clients at a time
C. No because it still blocks on recv
D. No because it now blocks somewhere else
E. What did we talk about on Friday again?
clientSockets = {}; 
while(1) {
    select on server socket and clientSockets
    if(server socket is ready)
        open connection and add it to clientSockets
    foreach clientSocket s ready {
        recv once on s
        if get data, echo it back
        else close s and remove from clientSockets
    }
}
clientSockets = {};  
while(1) {  
  select on server socket and clientSockets  
  if(server socket is ready)  
    open connection and add it to clientSockets  
  foreach clientSocket s ready {  
    recv once on s  
    if get data, echo it back  
    else close s  
  }  
}
clientSockets = {}; 
while(1) {
    select on server socket and clientSockets
    if(server socket is ready)
        open connection and add it to clientSockets
    foreach clientSocket s ready {
        recv *once* on s
        if get data, echo it back
        else close s
    }
}
Another alternative: polling

• Can tell blocking calls not to block
  – Ex: set recv’s last argument to MSG_DONTWAIT

• If the operation isn’t ready, it returns an error indicating that this was the problem
What is the most serious issue with polling?

A. Options to make non-blocking calls are non-standard
B. Never lets the process wait for something to happen
C. It increases network traffic
D. Something else
E. There isn’t any significant issue with it
What is the most serious issue with polling?

A. Options to make non-blocking calls are non-standard
B. Never lets the process wait for something to happen
C. It increases network traffic
D. Something else
E. There isn’t any significant issue with it
Bonus topic: Where do IP addresses come from?
Original scheme for granting IP addresses

• Organizations granted network in 3 categories:
  – Class A (addresses starting with 0)
    7 bits to specify network, 24 for host
  – Class B (addresses starting with 10)
    14 bits for network, 16 for host
  – Class C (addresses starting with 110)
    21 bits for network, 8 for host

• Other addresses for multicast or future use
What does Knox need?

A. Class A: >64K hosts, but ≤16M
B. Class B: >256 hosts, but ≤64K
C. Class C: ≤256 hosts
D. More than 16M hosts
E. Why does Knox need this silly Internet stuff?
How many Knox-sized (in terms of IP address needs) organizations are there in the world?

A. ≤128
B. >128, but ≤64K
C. >64K, but ≤2M
D. >2M
E. None of the above
XKCD “Map of the Internet”

http://xkcd.com/195/

Knox is in block 198
IPv4 addresses inefficiently allocated in blocks

- Eventual solution: IPv6 (get more addresses)

- 1\textsuperscript{st} solution: Allocate addrs more efficiently
  - Classless InterDomain Routing (CIDR)

- 2\textsuperscript{nd} solution: Use few addrs more efficiently
  - Network Address Translation (NAT)
• Assume traffic is TCP or UDP
• Replace external port number with index into table identifying internal host and port
• Deal with other protocols on case-by-case basis
Objections to NAT

• Violates model of IP (one addr per computer)
• Router needs to keep state of connections
• Breaks separation between network and transport protocol layers
• Some applications include IP addresses in the body of messages (e.g. ftp)

In general, a temporary and ugly hack that delays real solution
Comic!

http://xkcd.com/742/