COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

CSCE 313
Network Socket MP8

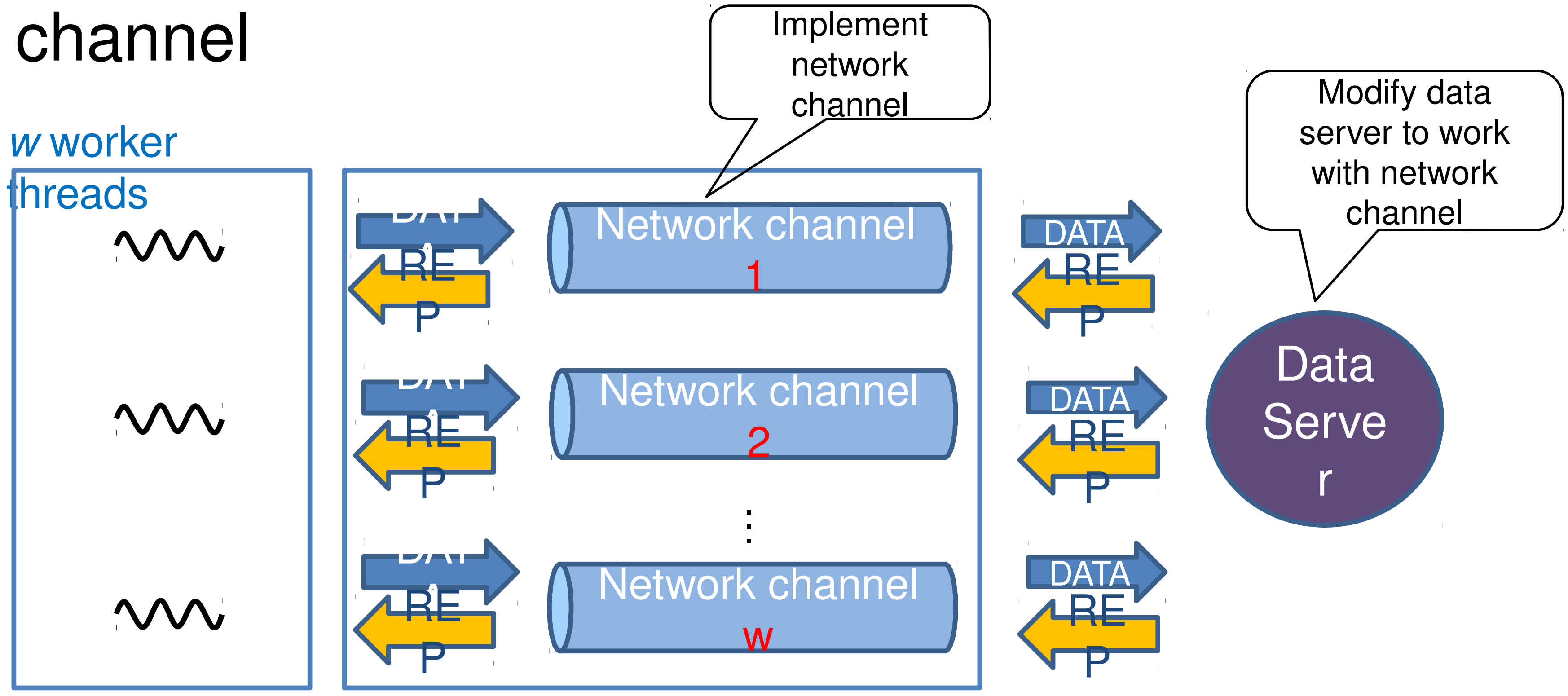
Wei Zhang

DUE: FRI MAY 5, 2017

Acknowledgment: Yi Cui, Wei Zhang, Prof. Tanzir Ahmed, CSCE-3

MP8

- Use network sockets for data communication between clients and servers
- Take MP7, change request channel to network channel



Your Task - Code

- Create your network request channel class to replace the original request channel that uses named pipes
  - NetworkRequestChannel.h, NetworkRequestChannel.cpp
- Modify workers to use network channels
  - Worker logic stays the same, but use new channels
  - Updated client.cpp
- Modify data server to use network channels
  - Updated dataserver.cpp
- Updated Makefile (as applicable)

Your Task - Report

- In addition, write a report with three key sections:
  - Performance Evaluation (especially relative to MP7)
  - Graph the response time versus the number of client requests
    - Plot multiple curves by varying the server backlog
  - Commentary on your program performance in context of the system you ran it on

Background: Network Socket

- Provides communication between different machines
- Need hostname/IP + port to uniquely identify the communication destination
- Can be used for inter-process communication within the same machine
  - Use 127.0.0.1 for local host IP
  - Port number must be some predefined number between clients and data servers
    - Use a large port number (e.g. 10000 or higher) to ensure you do not conflict with any well known ports

Background: How it works

- Data server side

```
s = socket()  // create socket
bind(s, port)  // bind socket to a specific port
listen(s) // start listening for incoming connections
for (;;) {
  new_s = accept(s) // when a new connection arrive, function returns a new socket
                                  // to be used for the new connection
  pthread_create(connection_handler)     // create a new thread to handle connection
}
```

- Client side

```
s = socket()  // create socket
connect(s, hostname, port)  // connect to data server
send requests via socket
```

- More details, see beej's guide
  - http://beej.us/guide/bgnet/output/html/multipage/index.html

System Call: socket()

- int s = socket(domain, type, protocol);
  - s: socket descriptor, an integer (like a file-descriptor)
  - domain: integer, communication domain
    - e.g., PF_INET (IPv4 protocol) – typically used
  - type: communication type
    - SOCK_STREAM: reliable, 2-way, connection-based service
    - SOCK_DGRAM: unreliable, connectionless,
    - other values: need root permission, rarely used, or obsolete
  - protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0

System Call: bind()

- associates and (can exclusively) reserves a port for use by the socket
- int status = bind(sockid, &addrport, size);
  - status: error status, = -1 if bind failed
  - sockid: integer, socket descriptor (i.e. the return value of socket(...))
  - addrport: struct sockaddr, the (IP) address and port of the machine
  - size: the size (in bytes) of the addrport structure

System Call: listen() and accept()

- int status = listen(sock, queuelen);
  - status: 0 if listening, -1 if error
  - sock: integer, socket descriptor
  - queuelen: integer, # of active clients that can “wait” for a connection
  - listen is **non-blocking**: returns immediately
- int s = accept(sock, &name, &namelen);
  - s: integer, the new socket (used for data-transfer)
  - sock: integer, the orig. socket (being listened on)
  - name: struct sockaddr, address of the client
  - namelen: sizeof(name)
  - accept is **blocking**: waits for connection before returning

System Call: connect()

- int status = connect(sock, &name, namelen);
  - status: 0 if successful connect, -1 otherwise
  - sock: integer, socket to be used in connection
  - name: struct sockaddr: address and port of server
  - namelen: integer, sizeof(name)
- connect is <u>**blocking**</u>

System Call: send() and recv()

- int count = send(sock, &buf, len, flags);
  - count: # bytes transmitted (-1 if error)
  - buf: char[], buffer to be transmitted
  - len: integer, length of buffer (in bytes) to transmit
  - flags: integer, special options, usually just 0
- int count = recv(sock, &buf, len, flags);
  - count: # bytes received (-1 if error)
  - buf: void[], stores received bytes
  - len: # bytes received
  - flags: integer, special options, usually just 0
- Calls are **blocking** [returns only after data is sent (to socket buf) / received]
- You can use write/read instead

System Call: close()

- When finished using a socket, the socket should be closed:
- status = close(s);
  - status: 0 if successful, -1 if error
  - s: the file descriptor (socket being closed)
- Closing a socket
  - closes a connection (for SOCK_STREAM)
  - frees up the port used by the socket