Week 7: Transport Layer

Lecturer: Tom Drummond

Semester 2, 2021

Connection Release

- Asymmetric Disconnection
  - Either party can issue a DISCONNECT, which results in DISCONNECT TPDU and transmissi ions
- Symmetric Disconnectia
  - Both parties issue DISCONNECT, closing only one direction at a time – allows flexibility to remain in receive mode

Connection Release (Cont.)

- Asymmetric vs Symmetric connection release types
- Asymmetric release may result in data loss hence symmetric re attractive
- Symmetric release works well where each process has a set amount of data to transmit and knows it has been sent

Generalizing the Connection Release Problem

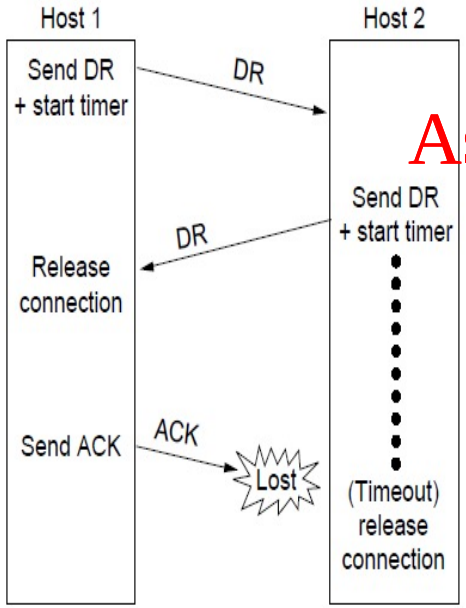
- How do we decide the importance of the last message? Is it essential or not?
- No protocol exists which can resolve this ambiguity – Two-army p f agreement

Strategies for Connection Release

- 3 way handshake
- Finite retry
- Timeouts
- Normal releas
  initiated by tra
  Host 1
  - DR=Disconnect Request
  - Both DRs are ACKed by the other side

Connection Release (Error Cases)

- Error cases are handled with timers and retransmission



Final ACK lost, Host 2 times out

Lost DR causes retransmissions

Extreme: Many lost DRs cause both hosts to timeout

Addressing

- Specification of **remote process to connect to** is required at application and transport layers
- Addressing in transport layer is typically done using **Trans s Points** (TSAPs)
  - on the Intern y referred to as a port (e.g. ***port*** 80)
- Addressing in the networ typically done using **Network Service Access Points** (NSAPs)
  - on the Internet, the concept of an NSAP is commonly interpreted as simply an ***IP address***

TSAPs, NSAPs and Transport Layer Connections Illustrated

Types of TSAP Allocation

1. Static
   - Well known services have standard allocated TSAPs/ports, which are embedded in OS
2. Directory Assistance – Port-mapper
   - A new servic portmapper, giving both its service na
3. Mediated
   - A process server intercepts inbound connections and spawns requested server and attaches inbound connection
   - cf. Unix /etc/(x)inetd

Programming using Sockets

- Sockets widely used for interconnections
  - “Berkeley" sockets are predominant in internet applications
  - Notion of “sockets” as transport endpoints
  - Like the simple set plus SOCKET, BIND, and ACCEPT

Recall Example Pseudo Code

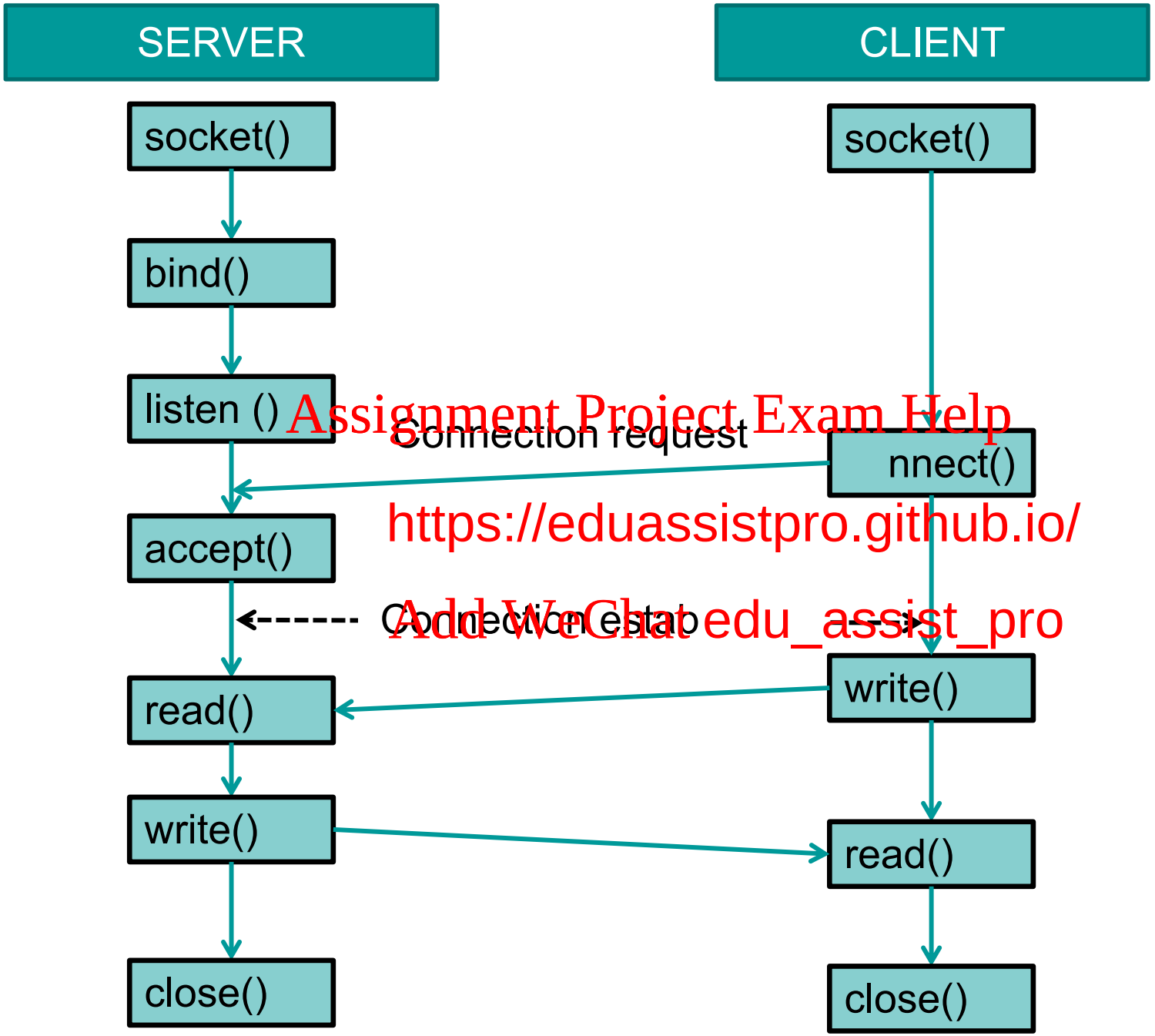
```
Socket A_Socket = createSocket("TCP");

connect(A_Socket, 128.255.16.0, 80);

send(A_so                          ;

disconnect(A_socket);
```


… there is also a server component for this client that runs on another host…

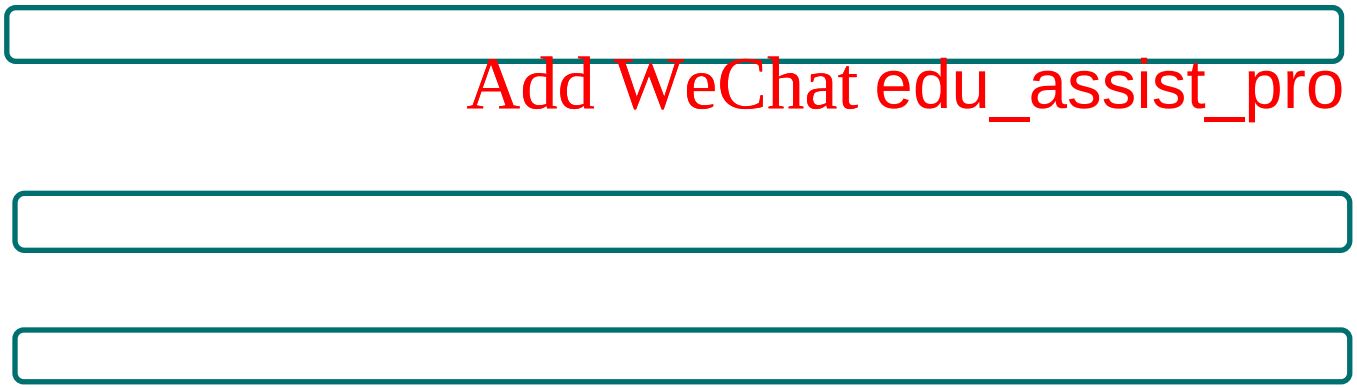


Let's Look at the Code from the book (in a specific language)

Example from the book has more details but the essence is the same… This is the case in most languages…

Socket Example – Server Side

Server code. . .



Assign address

Prepare for incoming connections

. . .

Server Code Contd

Block waiting for the next connection

ead (receive) quest

.....

The server can also create a new thread to handle the connection on the new socket and go back to waiting for the next connection on the original socket…

An Example on Multi-Threading

```
ServerSocket serverSocket = new ServerSocket([parameters]);

While (true) {
        Socket socket = serverSocket.accept();
        MultiThread                                    tiThreadMyServer();
        server.setM                                    eters]);
        server.setSocket(socket);
        new Thread(server).start();
    ....
```


- *(Code from OO Programming with Java; Chp. 14)*

Looking under the hood for Transport Layer Services…

- The **most basic** is actually connectionless:
  - Called: **User Datagram Protocol (UDP)**
  - Does **not add** functionality
  - TCP we just does the real-deal fo *liability*…
  - For UDP: Just remove connectio use it in a program
  - **UDP good for?:**
    - It is used for apps like video streaming/gaming regularly
  - **The reliability issue is left to?:**
    - the application layer… retransmission decisions as well as congestion control

New Code: UDP Client…

```
public static void main(String args[]) {
        ….
        Datagra                          = new
              Da
        mySocket.send([dat                s, etc
              parameters]);
        …
}
```

Server Side: UDP Example Contd

```
public static void main(String args[]) {
        ….
        DatagramSocket server = new
                Da                          );
        while (true) {
                server.receive([parameters]);
                …
        }}
```