Computer Network
Socket Programming Project

[吳孟勳] vincentwu007@cmlab.csie.ntu.edu.tw
[胡家愷] garyhu@cmlab.csie.ntu.edu.tw
[林方綺] sandy@cmlab.csie.ntu.edu.tw

Outline

- Project Description
- Tasks & Grading
- Phase 1
- Phase 2
- Requirements & Submission
- Introduction to Socket Programming

Project Description

- The project is a **real-time online chatroom** application that allows users to communicate through various media types, including **text, files, and live video streaming**. The system is designed to support seamless and **secure communication** in both private and public chat modes.

Tasks & Grading

- Phase 1 (40pts)
    - Basic Server-Client Communication (20pts)
    - Authentication Features (20pts)
- Phase 2 (65pts)
    - Multithread server (15pts)
    - Sending Chat Messages (20pts)
    - Message Encryption with OpenSSL (10pts)
    - Transfer files (10pts)
    - Audio/Video Streaming (10pts)
- Bonus (25pts)
    - Microphone and Webcam Integration (10pts)
    - GUI Interface (5~15pts)

Phase 1

Basic Server-Client Communication (20pts)

- Server (+10 pts)
  - Server can receive messages from the client.
  - Server can respond to client messages.
- Client (+10 pts)
  - Client can send messages to the server.
  - Client can receive responses from the server.

Authentication Features (20pts)

- User Registration (+7 pts)
- User Login (+7 pts)
  - You have to demo a case of login failure (e.g., the user doesn't exist).
- User Logout (+6 pts)
  - Can’t kill the process directly.(ex: ctrl+c, close the window)

Phase 2

Multithread server (15pts)

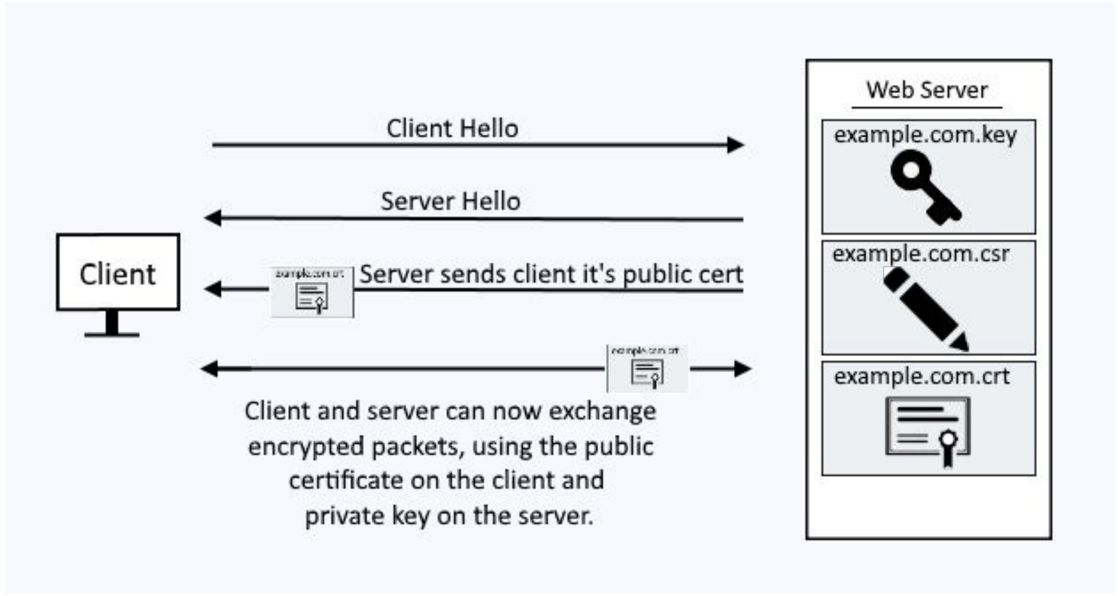
- Create a high-performance multithreaded server that can handle up to 10 concurrent connections using a worker pool approach.
- This server should be implemented using **POSIX threads (pthread) and worker pool design pattern**.
- Do not use fork() for process creation.

Sending Chat Messages (20pts)

- The system should support two sending modes: Relay and Direct Mode.
  - Relay mode: the client sends messages to another **online** client through the server.
  - Direct Mode (peer-to-peer): messages are sent directly from client to client without passing through the server.
- Hints
  - Client should create a listening socket to receive incoming messages.
  - Clients should inform the server about their listening port upon connection.
  - Clients should obtain other clients' IP addresses and ports from the server.
- Requirements
  - **Single-person team**: Implement either Relay Mode **or** Direct Mode.
  - **Two-person team**: Implement both Relay Mode **and** Direct Mode.

Message Encryption with OpenSSL (10pts)

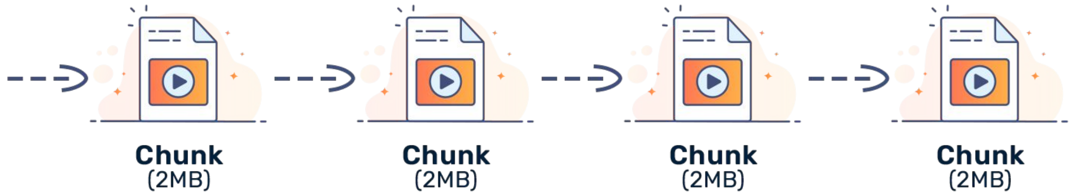
- Implement a secure communication system for both client-to-client and client-to-server interactions.


Web Server
example.com.key
example.com.csr
example.com.crt
Client Hello
Server Hello
Client
Server sends client it's public cert
Client and server can now exchange encrypted packets, using the public certificate on the client and private key on the server.

source: https://www.freekb.net/Article?id=1979

Transfer files (10pts)

- Implement a file transfer feature based on existing communication system.
- Requirements
  - Send and receive files in chunks.
  - Design and implement your own method for file selection on the sender's side and file saving on the receiver's side.
  - **Need encryption with OpenSSL**
  - You are only allowed to use standard library and POSIX library.



Audio/Video Streaming (10pts)

- Develop a frame-based streaming feature for audio or video files.
- Requirements
  - Sender: Implement file selection and frame-based transmission.
  - Receiver: Implement frame reception, reassembly, and **real-time** playback/display.
  - You are allowed to use any library.
  - **Encryption is optional.**
  - **Single-person team**: Implement either video **or** audio streaming.
  - **Two-person team**: Implement both video **and** audio streaming.

(Bonus) Microphone and Webcam Integration (10pts)

- Extend your audio/video streaming feature to capture and stream real-time audio from the computer's microphone and video from the webcam.
- You can implement either microphone integration or webcam integration.
- You are allowed to use any library.

(Bonus) GUI Interface (5~15pts)

- Develop a graphical user interface (GUI) for this entire chat system.
- You are allowed to use any library.
- We will grade the GUI based on its completeness and visual appeal.
- If you enhance the input/output interface of the terminal, you can earn up to an additional 5 points.

Requirements & Submission

Requirements

- Only C / C++ (Unix/Linux Socket Programming)
  - For windows user, you can use Win Socket or install WSL.
- Accepted Library for socket programming
  - #include <sys/socket.h>
  - #include <netinet/in.h>
  - #include <arpa/inet.h>
  - #include <winsock2.h>
- **No plagiarism**. Both the plagiarist and the original author will receive a score of 0 if plagiarism is detected.
- Team
  - Phase 1: Only individual submissions are allowed.
  - Phase 2: 1~2 people each team.

Submission

- Phase 1 (Deadline: **2024/11/15 23:59**)
  - **1 minute demo video**
  - Your code
  - README file
- Phase 2 (Deadline: **2024/12/27 23:59**)
  - **5 ~ 10 minutes video (including demo and code explanation)**
  - Your code
  - README file
- Your README file must include
  - Compilation instructions
  - Usage guide
  - Any additional information necessary to run your project

Submission

- Submit your code and YouTube Link on NTU COOL
- For both Phase 1 and Phase 2, submit a zip file named {student_number}.zip (e.g. b11902666.zip) containing:
  - b11902666/
    - README.md
    - code/
      - xxx.cpp
      - xxx.cpp
      - (other files or folders)
- If there is an incorrect file name or format, 5 points will be deducted.
- Late Submission Policy
  - Within 24 hours after the deadline: 70% of the original score
  - Beyond 24 hours: Submissions will not be accepted

FAQ

- 繳交格式一定要按照規定嗎？
  - 是，請將所有程式碼放到 code 資料夾底下，code 資料夾底下沒有規定排列方式
- 超過 100 分會以實拿分數計算嗎
  - 超過 100 分會以 100 分計算，但加分題可以提供你不同的拿分策略，或是挑戰更高的完成度。
- Demo 影片要搭配解說嗎
  - Phase 1 請搭配文字說明或是講解，Phase 2 請完整講解功能及實作方式
- Phase 1 只要處理一個client就好嗎？
  - 是
- Phase 1 當server process結束之後需要保留註冊者資訊嗎？
  - 沒有特別規定，可以依照同學的實作方式自行決定

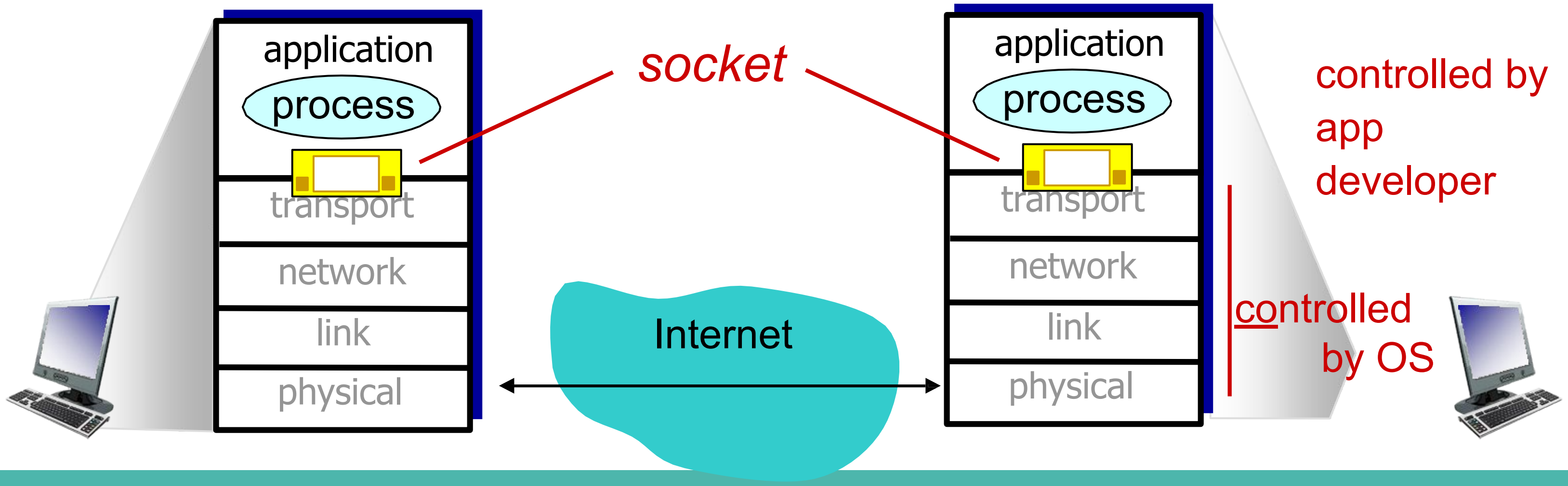
FAQ

- 可以假設 client 都有 public IP 嗎？
  - 可以假設 client 在相同的 Internet 底下，都可以透過 IP Address 互相通訊。
- 如果為兩人一組有需要在哪邊註明嗎？
  - 請在 README 以及影片中註明同組的所有同學們，作業繳交只需要由其中一位上傳即可。
- Phase 2 影片要有 code explanation，code explanation 要針對那些功能講解呢？
  - 請針對 Phase 2 新增的所有功能進行講解，需要投影程式碼的部分搭配講解，讓助教能 夠知道實作方式，不需要逐行講解，只要針對流程做大概的講解。

Introduction to Socket Programming

Socket

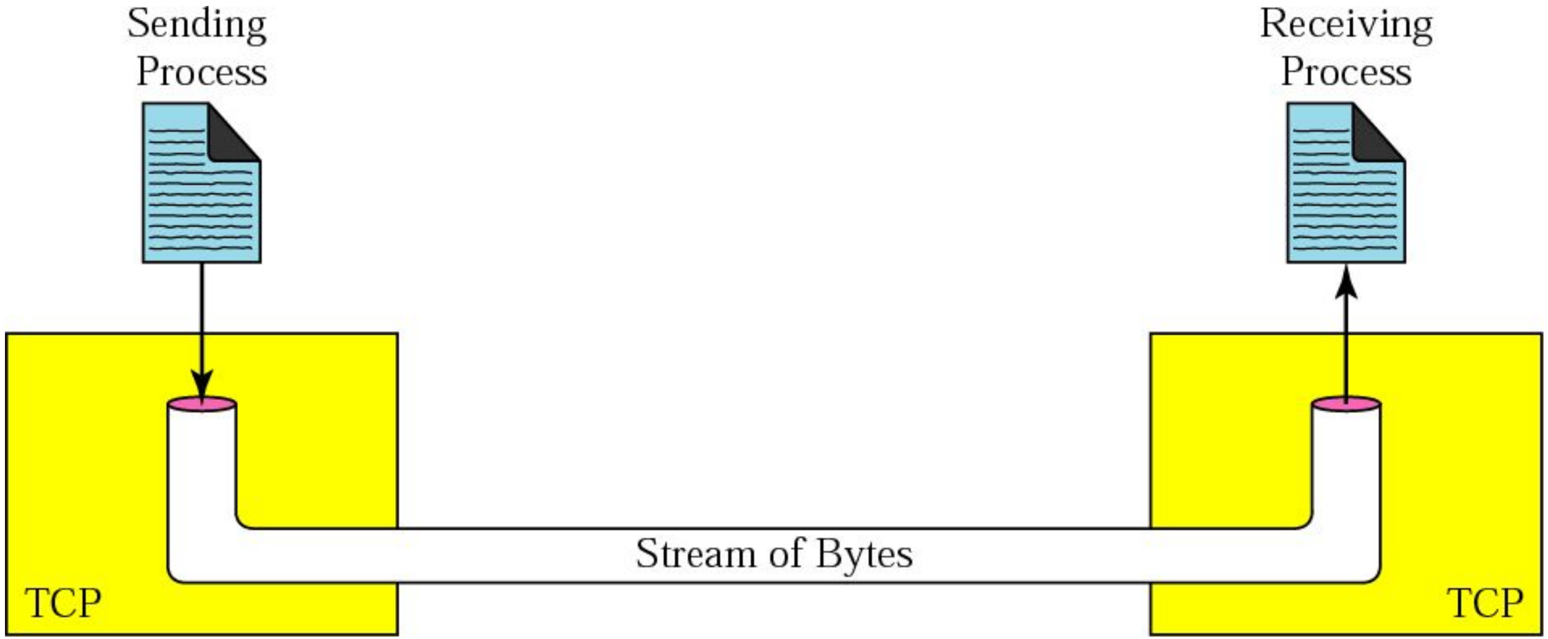
- Process sends/receives messages to/from its socket
- Socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



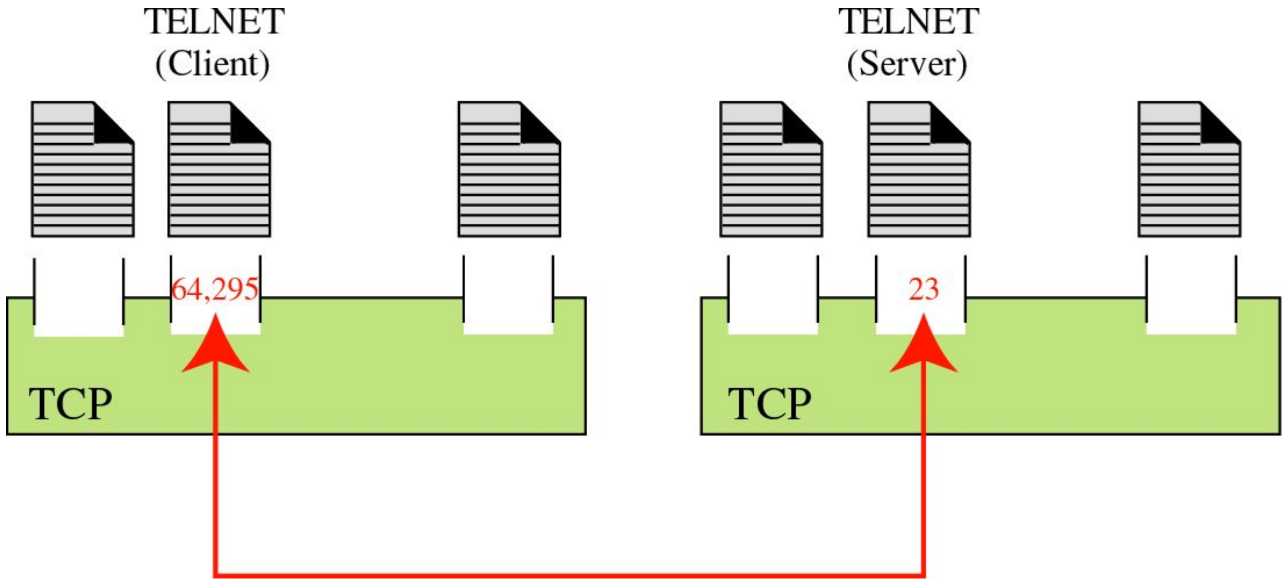
Socket Programming

- **Goal:** learn how to build client/server applications that communicate using sockets
- **Socket: door** between application process and end-end-transport protocol
- Two socket types for two transport services:
    - **UDP:** unreliable datagram
    - **TCP:** reliable, byte stream-oriented

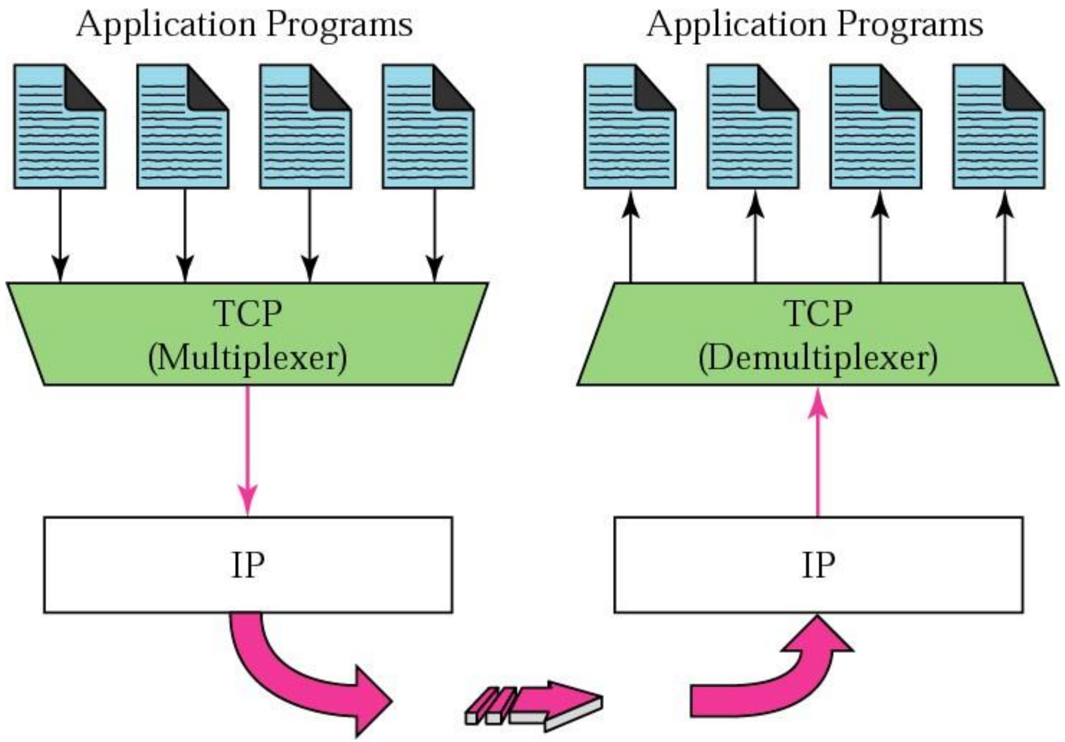
Stream Delivery



Port Numbers



Multiplexing and Demultiplexing



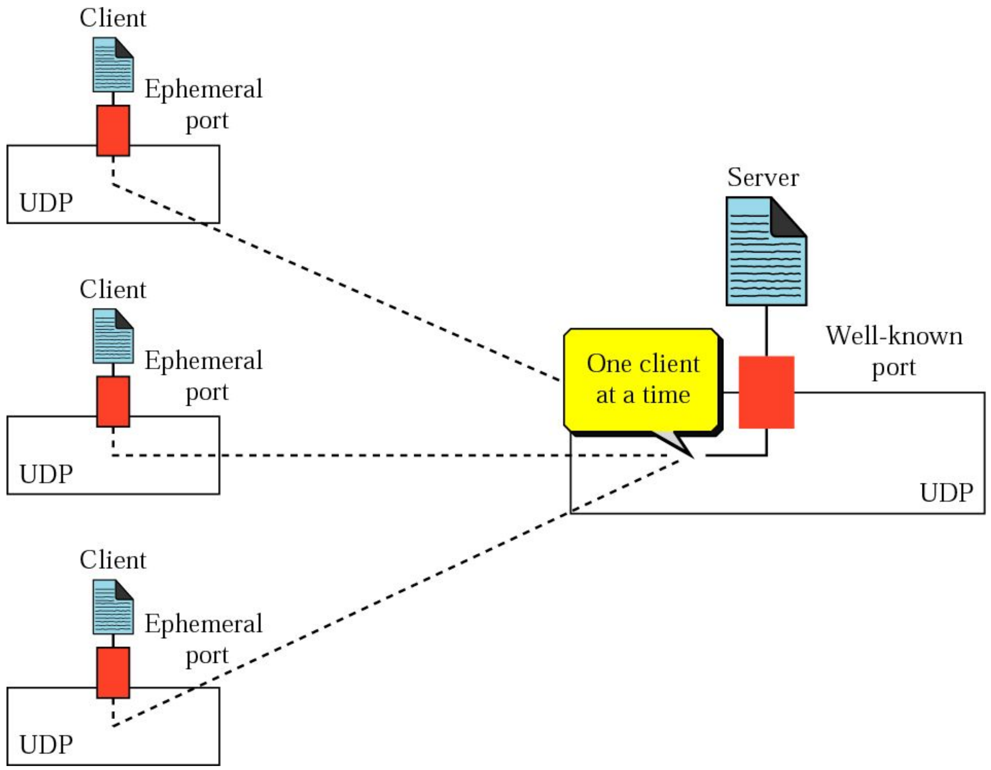
Socket Programming with UDP

- UDP: no connection between client & server
  - **no** handshaking before sending data
  - **sender** explicitly attaches **IP dest**ination address and **port** number to each packet
  - **receiver** extracts sender **IP addr**ess and **port** number from received packet
- UDP: transmitted data may be **lost** or received **out-of-order**
- Application viewpoint:
  - UDP provides **unreliable data transfer** of groups of bytes ("datagrams") between client and server

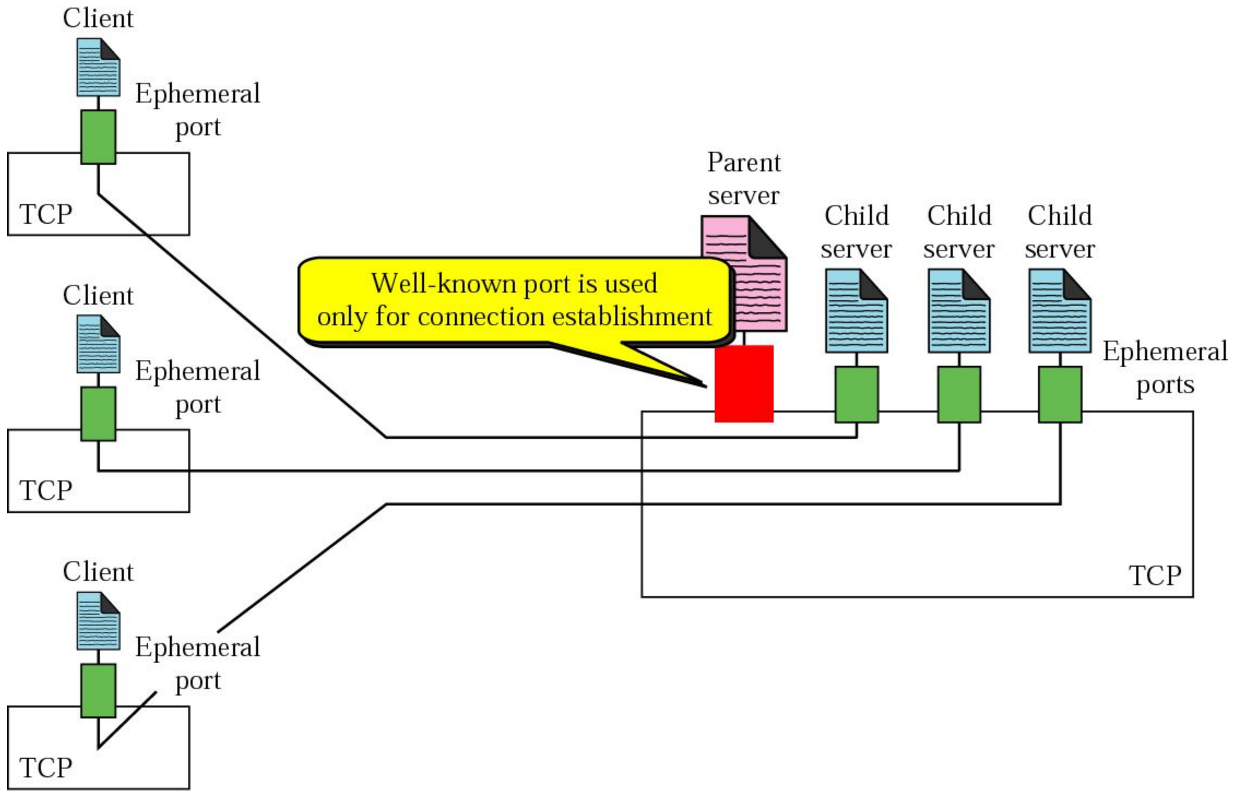
Socket programming with TCP

- client must contact server
  - server process **must first be running**
  - server must have created **socket** (door) that **welcomes client's** contact
- client contacts server by:
  - Creating **TCP socket**, specifying **IP addr**ess, **port** number of server process
  - When client creates a socket, a **TCP connection is established** between client and server
- when contacted by client, server TCP creates new socket for server process to communicate with that particular client
  - allows server to talk **with multiple clients**
  - **source port** number & **source IP** used to distinguish clients (more in Chap 3)
- Application viewpoint:
  - TCP provides **reliable**, **in-order byte-stream** transfer ("pipe") between client and server

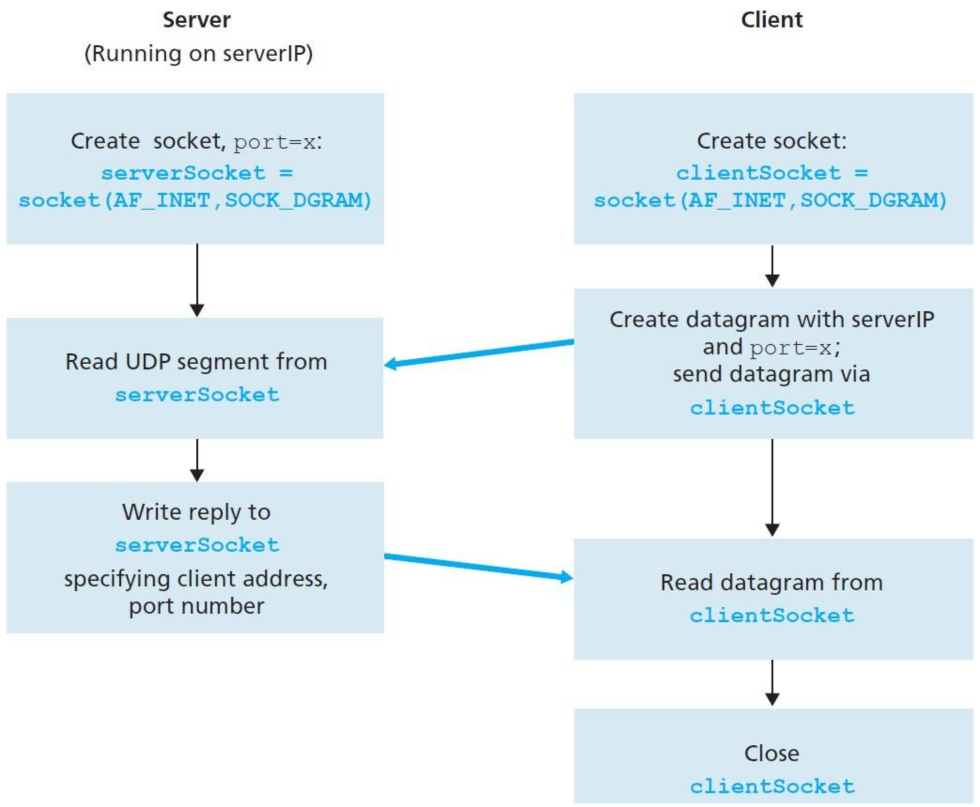
Connectionless iterative server (UDP)



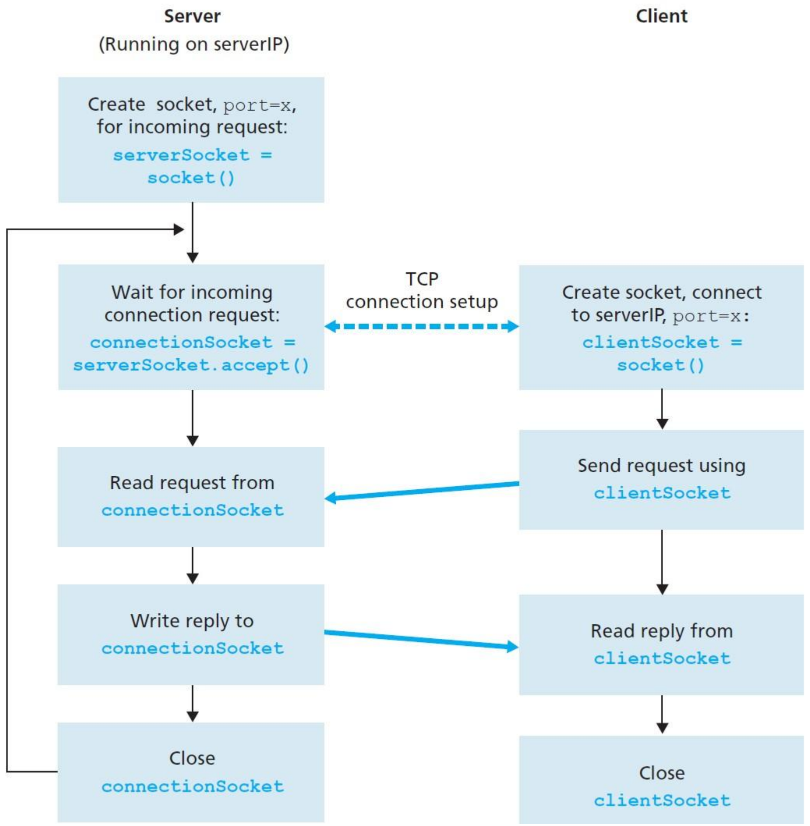
Connection-oriented concurrent server (TCP)



Client/server socket interaction: UDP

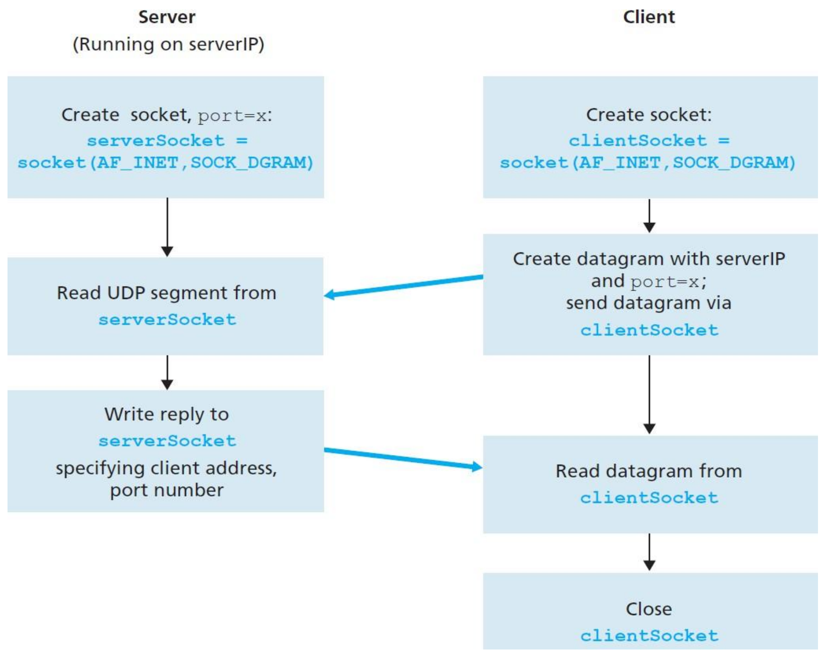


Client/server socket interaction: TCP

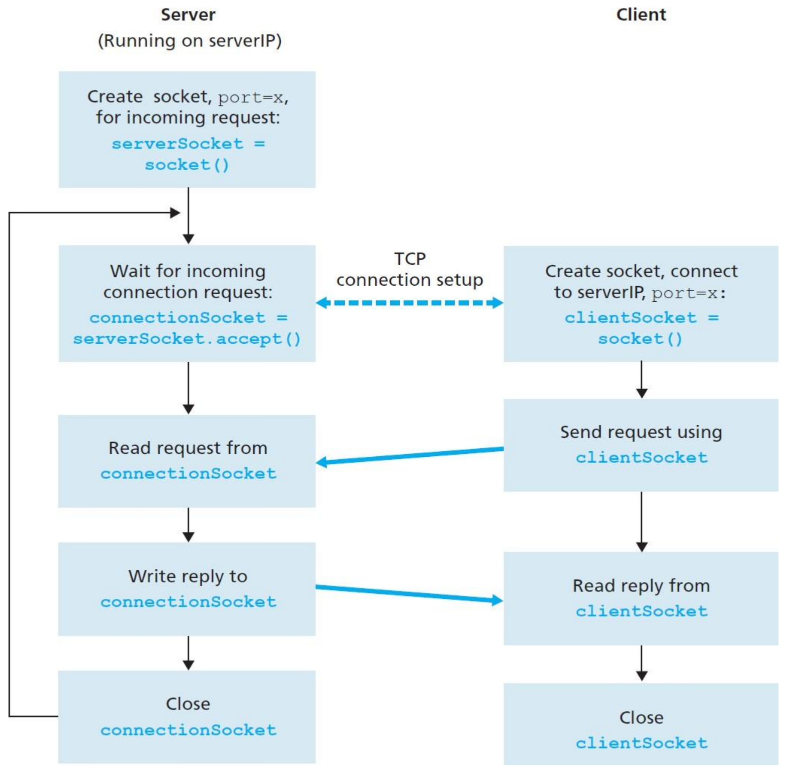

Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Send request using clientSocket
Read request from connectionSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

Client/server socket interaction

UDP


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket = socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket = socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP and port=x; send datagram via clientSocket
Read UDP segment from serverSocket
Write reply to serverSocket specifying client address, port number
Read datagram from clientSocket
Close clientSocket

TCP


Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Send request using clientSocket
Read request from connectionSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket