

# FTP

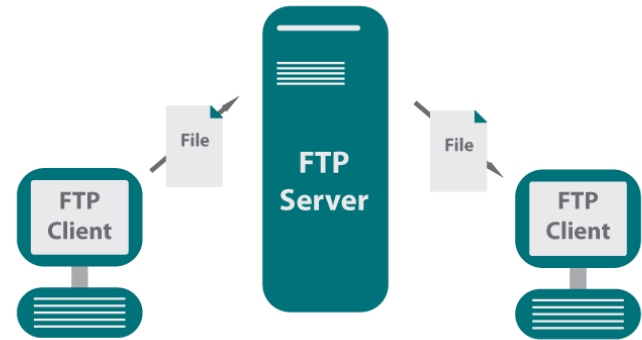
Computer Network  
Project 1

# What is FTP?

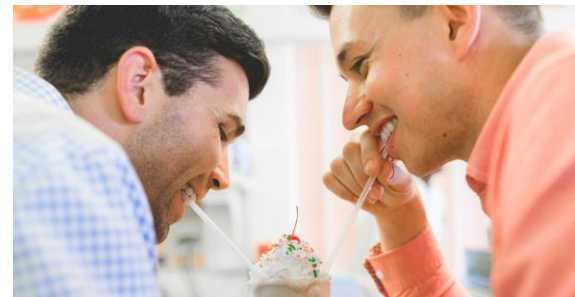
- Application layer protocols: SSH, P2P, FTP, etc.
- A protocol used for file download or upload between server and client.



SSH



P2P



## How to implement an FTP?

1. Open computer;
2. Write code;
3. Run code;

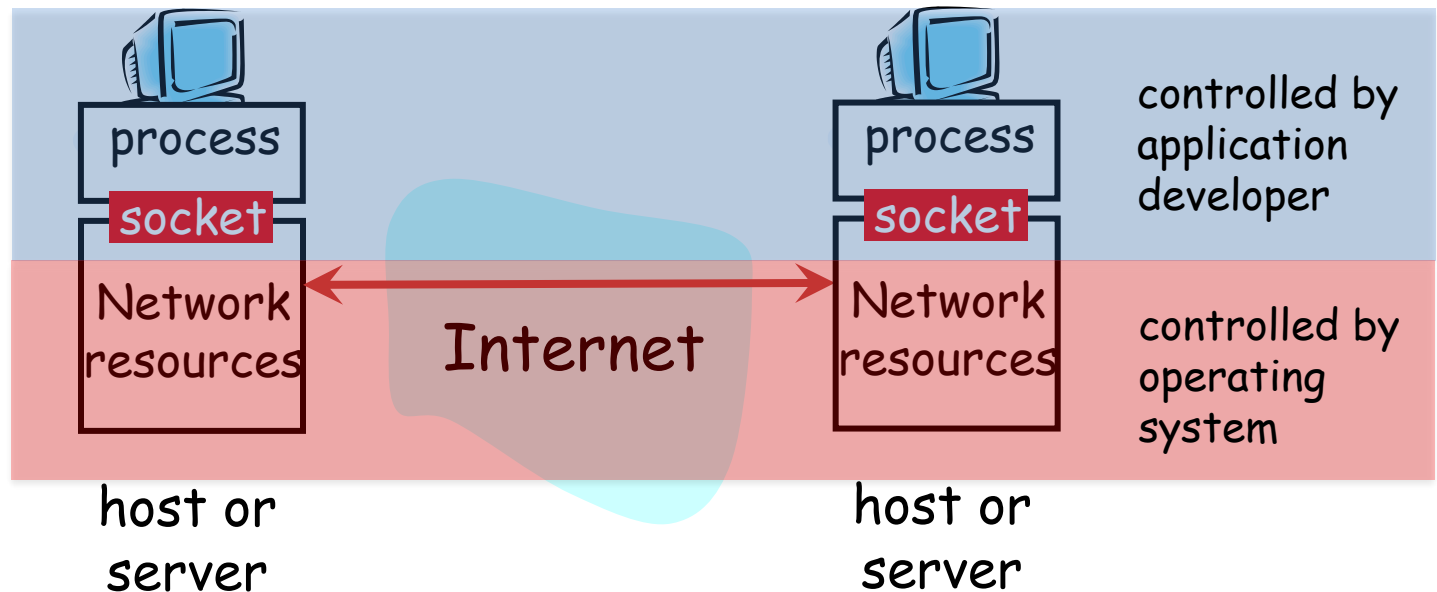
Is that so?



# Socket Programming

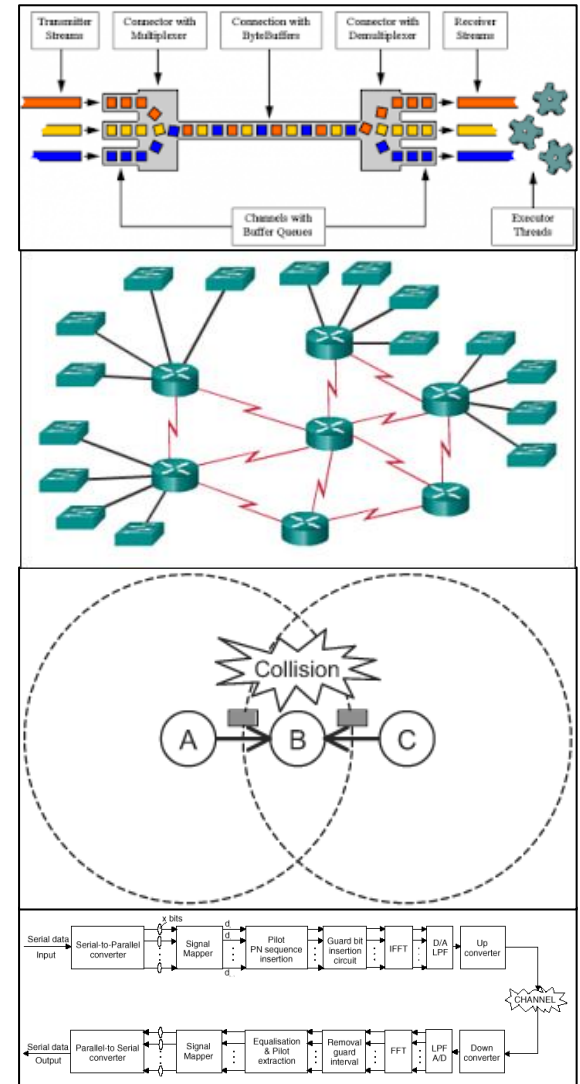
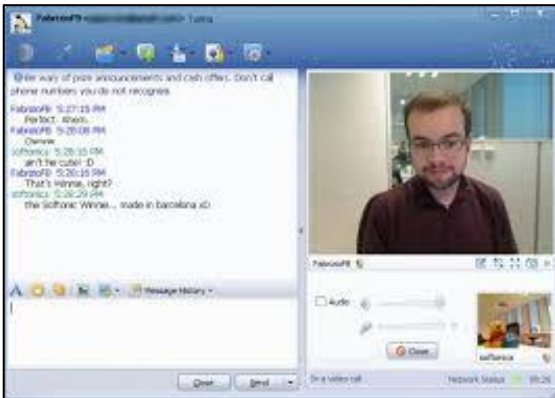
# What is Socket?

- Socket is an API to use internet resources.
- **Host-local, application-created, OS-controlled** interface.



# Why do we need socket?

- Layering



# Socket Identification

Mail	Call	Socket
Your Address	Your Phone Num.	Local Socket Address
Her Address	Her Phone Num.	Remote Socket Address
Envelop	Radio	Protocol

□ □ □ □ □ □  
收信人邮编

收信人地址

收信人姓名

寄信人地址, 姓名

寄信人邮编

邮政编码:



Socket?

# Address

- The address must identify the sender/receiver.
  - Identification of the host - IP Address
    - IPv4: 32bit, X.X.X.X, X:8 bit
    - IPv6: 128bit, Y:Y:Y:Y:Y:Y:Y:Y, Y:16 bit
  - Identification of the process - Port
    - 0~65535
- The five tuples to specify a socket:
  - (SrcIP, SrcPort, DstIP, DstPort, Protocol)



# Port Usage

- Popular applications have “well-known ports”.
  - By convention, between 0 and 1023; privileged
  - E.g., port 80 for HTTP, 25/465 for SMTP, 21 for FTP
  - For more, refer to Wikipedia.
- Custom client gets an unused “ephemeral” port.
  - By convention, between 1024 and 65535.

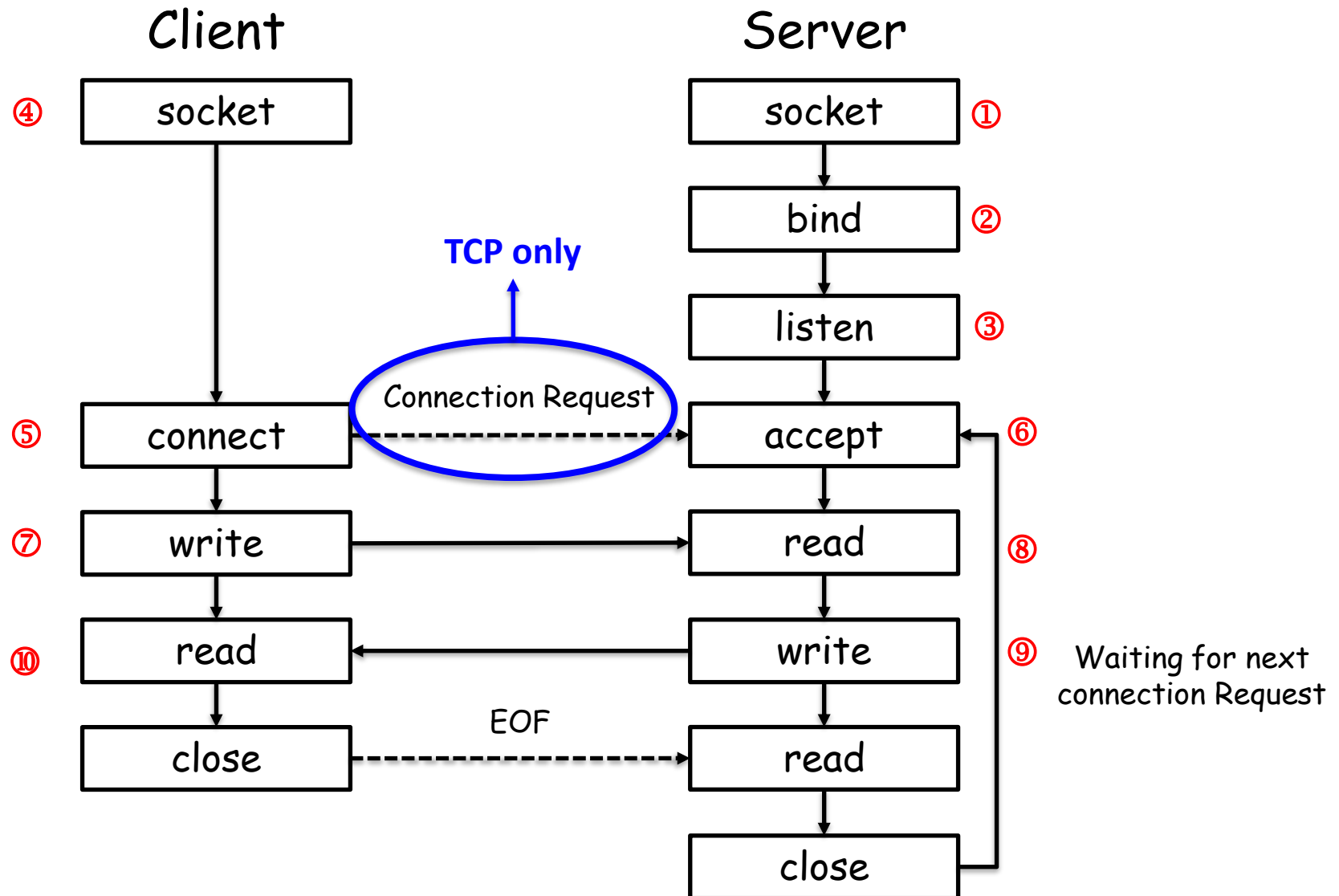
```
[root@esonjohn ~]# netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp      0      0 0.0.0.0:21             0.0.0.0:*               LISTEN      790/vsftpd
tcp      0      0 0.0.0.0:25             0.0.0.0:*               LISTEN      807/exim
tcp      0      0 0.0.0.0:1723           0.0.0.0:*               LISTEN      847/pptpd
tcp      0      0 0.0.0.0:587            0.0.0.0:*               LISTEN      807/exim
tcp      0      0 0.0.0.0:80             0.0.0.0:*               LISTEN      835/nginx
tcp      0      0 0.0.0.0:465            0.0.0.0:*               LISTEN      807/exim
tcp      0      0 0.0.0.0:28883          0.0.0.0:*               LISTEN      776/sshd
```

# Transport Layer Protocol

- Socket at Transport Layer.
  - TCP
    - Connection-oriented protocol
    - Reliable, ordered, heavyweight and streaming
  - UDP
    - Connectionless protocol
    - Unreliable, not ordered, lightweight and datagrams

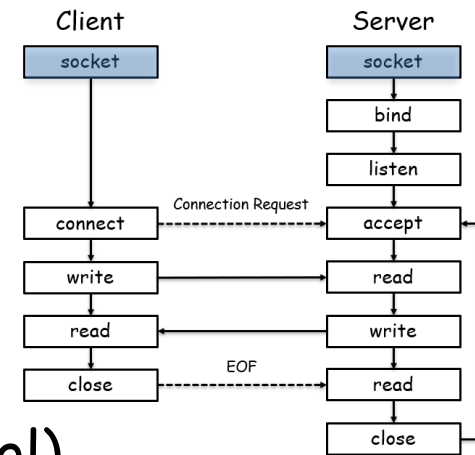


# Socket Programming in C



# API Functions

- Create a socket
  - `int socket(int domain, int type, int protocol)`
    - Domain: ipv4/ipv6
    - Type: `SOCK_STREAM/SOCK_SEQPACKET/...`
    - Protocol: `TCP/UDP/...`



# API Functions

- Server listening

- Bind some port

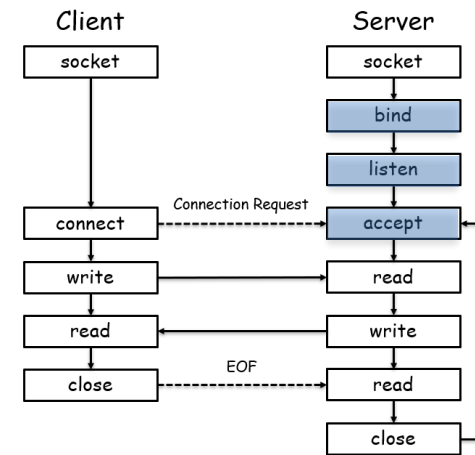
- `int bind (int sockfd, struct sockaddr *my_addr, int addrlen)`
      - sockfd: socket handler
      - my\_addr: local listening address
      - addrlen: my\_addr length

- Define how many connections can be pending

- `int listen(int sockfd, int backlog)`
      - sockfd: socket handler
      - backlog: maximum connections

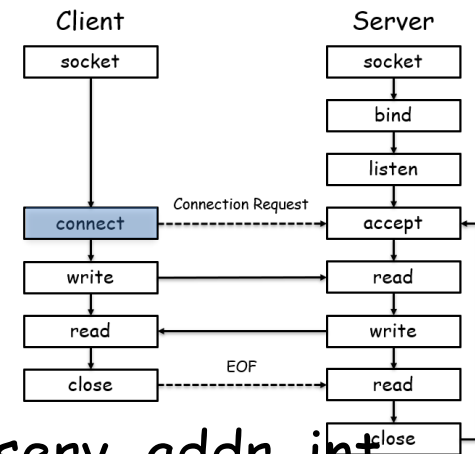
- Accept a new connection from a client

- `int accept(int sockfd, struct sockaddr *addr, int *addrlen)`
      - sockfd: socket handler
      - addr: client address
      - addrlen: addr length

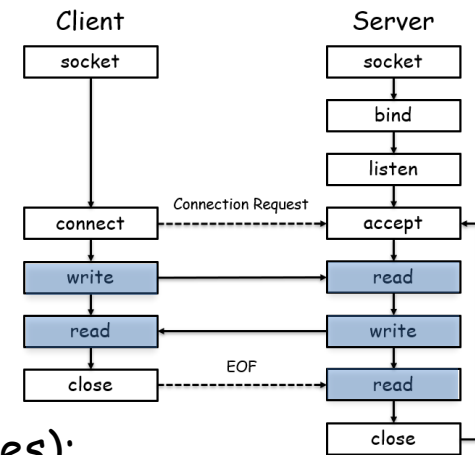


# API Functions

- Establish a connection
  - `int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)`
    - `sockfd`: socket handler
    - `serv_addr`: server address
    - `addrlen`: `serv_addr` length



# API Functions



- For TCP, Send Data

- `ssize_t write(int sockfd, const void *buf, size_t nbytes);`
- `ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);`
  - sockfd: socket handler
  - buf: data buffer
  - nbytes: write/send data length
  - flags: block/nonblock/...

- For TCP, Receive Data

- `ssize_t read(int sockfd, void *buff, size_t nbytes);`
- `ssize_t recv(int sockfd, void *buff, size_t nbytes, int flags);`

- For UDP, use

- `ssize_t sendto (int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr *dest_addr, int addrlen);`
- `ssize_t recvfrom (int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr *src_addr, int addrlen);`

# API Functions

- Byte Ordering

- unsigned long int htonl(unsigned long int hostlong)
- unsigned short int htons(unsigned short int hostshort)
- unsigned long int ntohl(unsigned long int netlong)
- unsigned short int ntohs(unsigned short int netshort)

- IP Addresses

- int inet\_aton(const char \*cp, struct in\_addr \*inp)
- char \*inet\_ntoa(struct in\_addr in);



# Other Tools

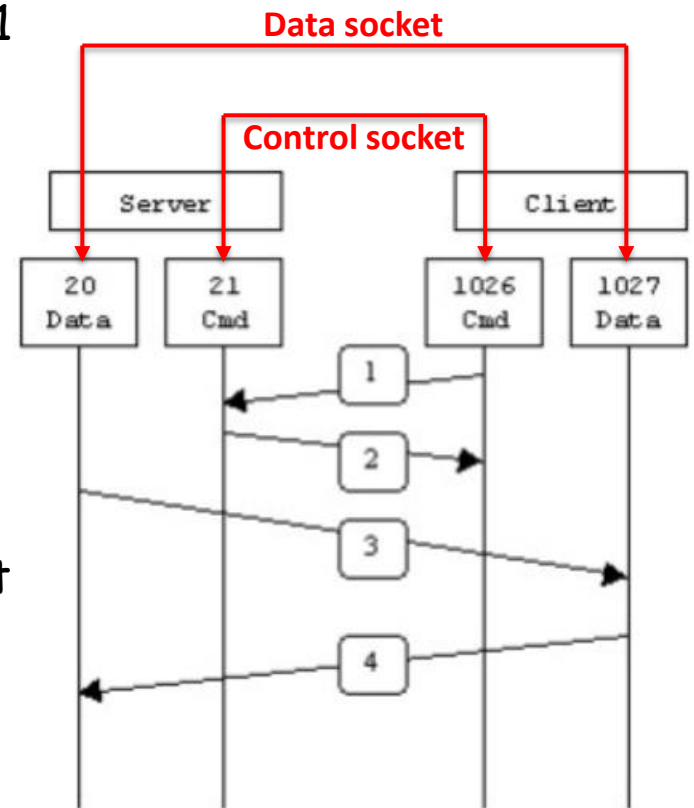
- How to check if particular port is listening
  - Windows - use netstat
    - netstat -an
  - Linux - use nmap
    - nmap -sT -O localhost
- Status code of bind function indicates port usage
- Not knowing what exactly gets transmitted on the wire
  - Use tcpdump or Ethereal ([www.ethereal.com](http://www.ethereal.com))
- Check RFC if in doubt about protocols.
  - RFC 959

# Commands in FTP

- USER, PASS
  - login to server
- RETR
  - download file from server
- STOR
  - upload file to server
- PORT, PASV
  - specify active/passive mode
- MKD, CWD, PWD, LIST
  - manipulate directories on server
- ...(rfc 959)

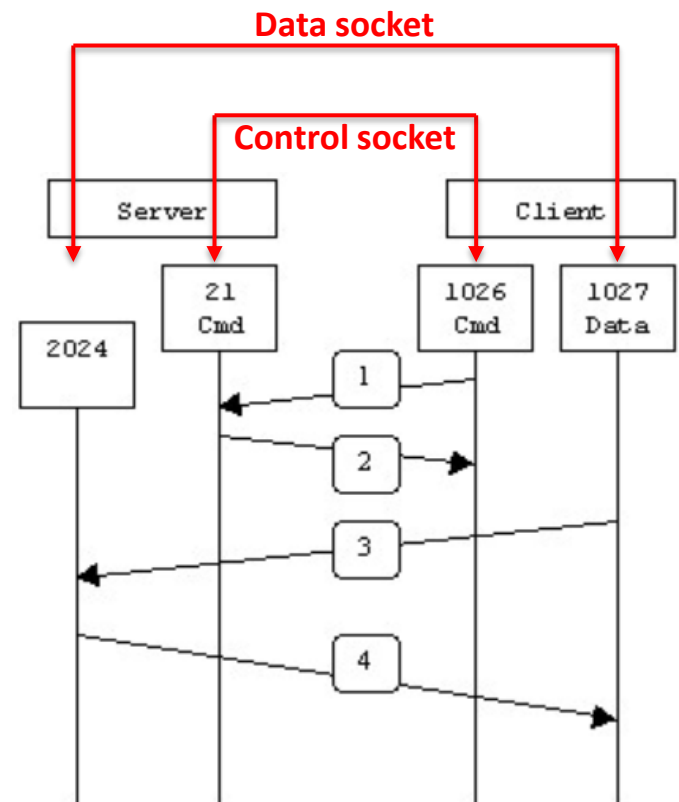
# PORT Mode of FTP

- PORT(Active Model)
  - Server monitor default control port 21
  - Client request control port N1 (1026) from OS
  - Client request data port N2 (1027) from OS and notify Server
  - Server setup connection between local 20 port and client N2 (1027) port
  - Data streams between server 20 port and client N2 (1027) port

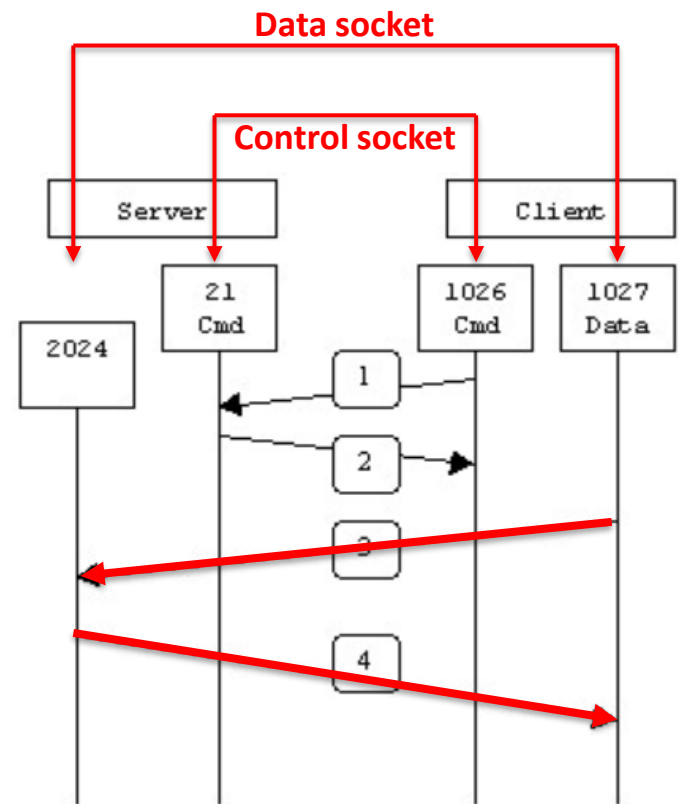
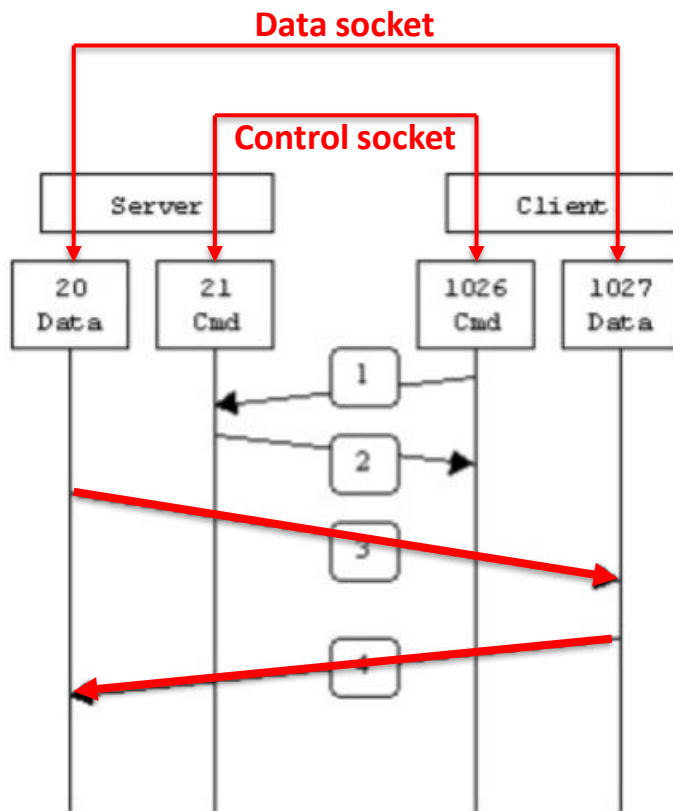


# PASV Mode of FTP

- PASV(Passive Model)
  - Server monitor default control port 21
  - Client request control port N1 (1026) from OS
  - Client request data connection from Server on N1 (1026)
  - Server request data port N2 (2024) from OS and notify Client
  - Client request data port N3 from OS
  - Client setup connection between local port N3 and server port N2
  - Data streams between server port N2 and client port N3



# PORT vs. PASV



# Project Description

- Goal
  - Learn to setup communications between PCs with socket.
  - To be familiar with FTP protocol;
  - To be familiar with network protocol stack;
  - Learn to build client/server applications that communicate using sockets.

# Task1: UDP Programming

- Read the sample code
  - Choose any language you are familiar with(C, C#, Java, node, Python).
- Modify the UDP Server and Client According to the homework guide.
- Questions:
  - How to write a chat programming (two clients chat with each other) with UDP?

## Task2: Implementing FTP Server

- Must use the Berkeley Socket API (Linux) and write your server in C;
- Must serve files from a **designated directory** on your system to clients making requests on a designated TCP port;
- Must handle USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, RMD commands;
- Handling **invalid input** reasonably and generating defensible error codes;



# Task2: Implementing FTP Server

- May not use any libraries containing code specifically designed to implement FTP functionality (coding at socket level);
- Support integral **large file** transmission;
- Support connections from **multiple** clients;
- **Optional**
  - (5%) **Resume** transmission after connection terminated;
  - (5%) File transmission **without blocking** the server;

# Task3: Implementing FTP Client

- You can write your client in your **favorite** programming language;
- Must handle USER, PASS, RETR, STOR, QUIT, SYST, TYPE, PORT, PASV, MKD, CWD, PWD, LIST, RMD commands;
- Able to log in a provided **commercial** FTP server and download/upload files;
- Support integral **large** file transmission;

# Task3: Implementing FTP Client

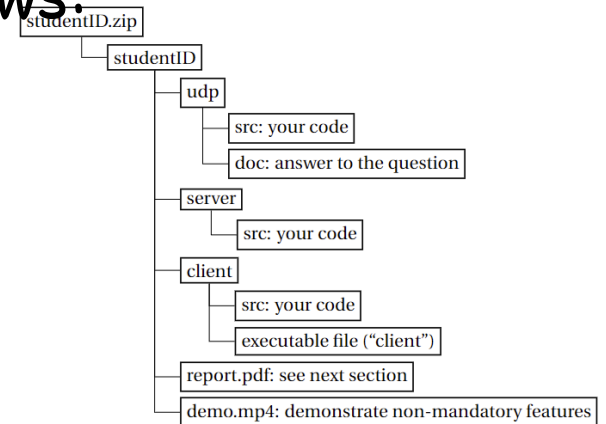
- May not use any libraries containing code specifically designed to implement FTP functionality (coding at socket level);
- **Optional**
  - (5%) **Resume** transmission after connection terminated;
  - (5%) Friendly *GUI*;
  - (5%) File transmission **without blocking** the *GUI*;

# Some Important Issues

- Grading
  - UDP Programming (10%)
  - Implementation of FTP (70%+10%)
    - Server (40%)
    - Client (30%)
    - Optional (5%+5%)
  - Project Report (20%)
- Individual work.

# Some Important Issues

- Submission
  - Source code for udp, server/client;
  - Executable file named "client";
  - **Video** to demonstrate non-mandatory features;
  - Report: no longer than **THREE** pages;
  - Files are to be organized as follows:



- Evaluation
  - Auto-grading program for FTP server & client.

# Some Important Issues

- Important Date
  - Start up: today (2025.10.11.)
  - Due date: 4 weeks later (2025.11.9.)
- Cheating will be punished.
  - Repetition rate > 20%
  - 0 score in the project

# 补交规则

- 正常提交: 在 DDL 前提交作业正常计分
- 最迟提交期限和惩罚
  - 每次作业最迟在 DDL 后一周 (7天) 内提交, 超出此期限**一律拒收**
  - 未超出此拒收期限的迟交作业得分\*0.8.
- 宽限期: 全部大作业共享 7 天宽限期。迟交**累计不超过**该期限的免于扣分惩罚

# 补交规则

## • 例子

- 第一次大作业按时提交，第二次大作业未在最迟补交期限前（DDL后7天）提交 → 第一次大作业正常计分，第二次大作业不予补交
- 第一次大作业迟交3天，第二次大作业迟交4天 → 累计迟交7天，未超过宽限期，两次大作业均正常计分
- 第一次大作业迟交7天，第二次大作业迟交1天 → 第一次大作业正常计分；第二次大作业累计超过宽限期，仅得80%
- 第一次大作业迟交7天，第二次大作业按时提交 → 累计迟交7天，未超过宽限期，两次大作业均正常计分



# Problems Emerged

- Improper use of control/data socket(list, pwd)
- Illegal design of command (PORT/PASV)
- Fails to connect to standard FTP server
  - Follow RFC protocol
- Security Issue
  - cd ..
- String Issue (\0)
- Exceptions
  - Socket exception
  - I/O exception

## How to implement an FTP?

1. Read RFC959 document;
2. Coding ON YOUR OWN;
3. Debugging;

# Previous Outstanding Project

## Student A

### 1. 实验环境搭建

- FTP server: Ubuntu20.04(WSL) + C(GNU C11) + CLion
- FTP Client: Windows10 & macOS + C++11&Qt 5.12 + Qt Creator

### 2. FTP server

#### 2.1 实现的命令与对应的错误处理

命令与参数	功能说明	错误处理
USER <username>	只接受“anonymous”这一个用户名。永远发送给用户 331 请求填写密码。	收到错误用户名在服务器内部改变状态为 WRONGUSER, 在 PASS 命令中处理
PASS <password>	用户名与密码为密码。正常时发送 230。	若用户名填写错误, 发送 530 表示拒绝。若该请求不是紧接在 USER 后发送的, 发送 503 表示拒绝。
SYST	告知客户端系统信息。正常时发送 215。	若请求中带有别的参数, 以 500 告知客户端格式错误。
TYPE <type>	仅接受“Type I”, 正常时发送 200。	以 504 应对请求非 TYPE I, 表示该参数不能处理。
PORT<h1,h2,h3,h4,p1,p2>	建立一个 socket 准备用于连接 (不立即连接)。正常情况发送 200。	未登录以 530 回复; 不能解析 IP 地址的情况以 501 回复; 以 425 应对创建 socket 失败的情况。
PASV	在一个随机端口开启监听并告知客户端, 等待客户端连接。正常情况发送 227 并告知客户端要连接的 IP 和端口。	以 530 回复未登录; 以 425 应对创建 socket 失败或客户端没有连接上的情况。
RETR <filename> STOR <filename>	根据两种连接模式确认好连接后进行文件的传输。确认连接后发送 150 告知客户端传输开始。开启新的线程进行文件传输, 传输成功以 226 告知客户端。	如果并不是两种连接状态之一, 直接发送 425 表示没有建立连接。在文件传输进程里因管道破坏失败则发送 426。打不开文件发送 451。
QUIT	关闭所有该用户连接, 发送 221	
LIST [path]	在主机上使用 ls -l 命令列出对应文件夹或文件的信息并发送给客户端。	如果给定的路径试图访问根目录的父目录, 或者找不到该路径, 发送 550。连接错误的处理方式同 RETR
MKD <path>	创建文件夹。	如果给定的路径试图访问根目录的父目录发送 550。
PWD	告知客户端当前工作目录。	
CWD <path>	把工作目录切换为指定目录。	如果给定的路径试图访问根目录的父目录或者找不到该路径, 发送 550。
RNFR <filename>	准备重命名指定文件。回复 350	文件不存在则发 450, 文件在根

RNTO <filename>	告知客户端可以重命名。 重命名为指定文件名。回复 250 告知客户端重命名成功。	目录外发 550。 没有事先发 RNFR 则以 503 回复。文件在根目录外回复 550, 重命名失败回复 553。
-----------------	--	---

#### 2.2 实现方式

将所有功能分为几个模块, 实现在了对应名字的文件中。utils 模块中定义了核心的结构体 User 和枚举类型的 UserState。User 中包含了每个连接进来的用户的基本信息, 具体参见 utils.h 中的注释。还包含了其他模块中要用到的一些工具函数, 如解析 IP 地址。

IO 模块中实现了与客户端数据的传输, 包括了四个函数, 分别对应发送消息、接收消息、发送文件、接收文件。

command 模块中对所有的命令处理方式进行了实现, 该模块中的所有函数都有着统一的函数签名。

core 模块中是服务器开启的核心逻辑, 包括初始化服务器, 每个连接的处理过程, 和根据用户请求调用不同命令的函数。

server.c 中只有调用初始化服务器的 main 函数。

服务器在 init\_server 函数中完成一系列初始化后会持续监听连接请求, 在连接数没有达到最大时接受请求并为每个连接开启新的线程。在这个线程中执行 main\_process 函数来处理客户端的请求。有一个 while 循环不断地处理请求和接收请求。

消息发送、消息接收函数以及所有实现命令的函数都有统一的返回风格, 以返回 0 作为操作成功的标志, 返回非 0 数表示运行错误。大部分命令都需要给用户发出回复, 直接采用 return send\_message() 的方式进行。如果发送消息成功了则会返回 0 表示命令运行成功, 所发送失败则说明客户端已经主动断开连接, 此时返回一个非 0 数。main\_process 中的循环会终止, 该用户被释放, 对应线程也会结束。

文件的传输都会开启新的线程, 不会影响接收客户端请求。若收到退出的请求会关闭传输中的文件描述符并结束线程。

用户的结构体中储存用户在服务器上的工作目录, 而服务器的根目录用全局变量 rootDir 存储, 在初始化时从参数中读取根目录, 默认为/tmp。

### 3. FTP client

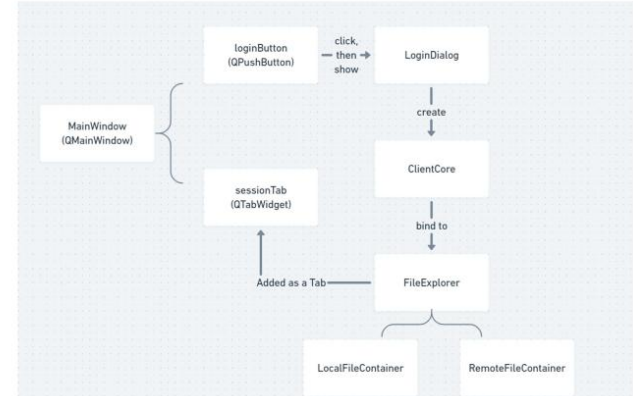
#### 3.1 功能实现与模块划分

客户端核心的功能实现在类 ClientCore 中, 该类继承 QObject, 有发出各个命令的函数与处理服务器回复、实现文件传输功能的函数。每次完成一项命令操作都会发出信号。

展示文件的类为 FileExplorer, 该类绑定一个 ClientCore, 调用 ClientCore 的函数进行对服务器的请求, 根据 ClientCore 的信号展示相关信息给用户。该类的界面包含有显示本地文件和远端文件的区域, 分别为 LocalFileContainer 和 RemoteFileContainer。

LocalFileContainer 和 RemoteFileContainer 继承自 FileContainer, FileContainer 继承自 QListWidget, 该类可以方便地以列表形式展示文件, 继承该类并重写一些函数是为了实现文件拖拽上传的操作。

主窗口为 MainWindow, 窗口包含一个 QTabWidget 来存放多个会话对应的文件浏览器。左上角的 New Session 按钮点击后会弹出登录框, 输入 IP 地址、端口、用户名与密码后就能进行登录。整体结构大致如下图:



#### 3.2 交互逻辑

提供了与常见的图形界面文件系统相似的操作方式:

1. 拖拽文件进行上传或下载
2. 点击路径栏直接进行路径切换, 或双击路径栏输入路径进行切换
3. 双击进入文件夹; 右键点击弹出操作菜单。
4. 鼠标放在远程文件上时用 ToolTip 显示更完整的文件信息。

在主要界面的左侧显示服务器的返回信息, 下方显示提示, 并提供了刷新远程文件夹和重新连接的按钮(重新连接按钮只有在检测到连接已断开时会启用)。

在进行文件传输时也可以在各个会话之间切换, 并对本地文件夹进行操作。但是不要对远程文件进行操作, 因为每次操作后都会发送 LIST 请求刷新文件夹, 这可能会导致文件传输中断。

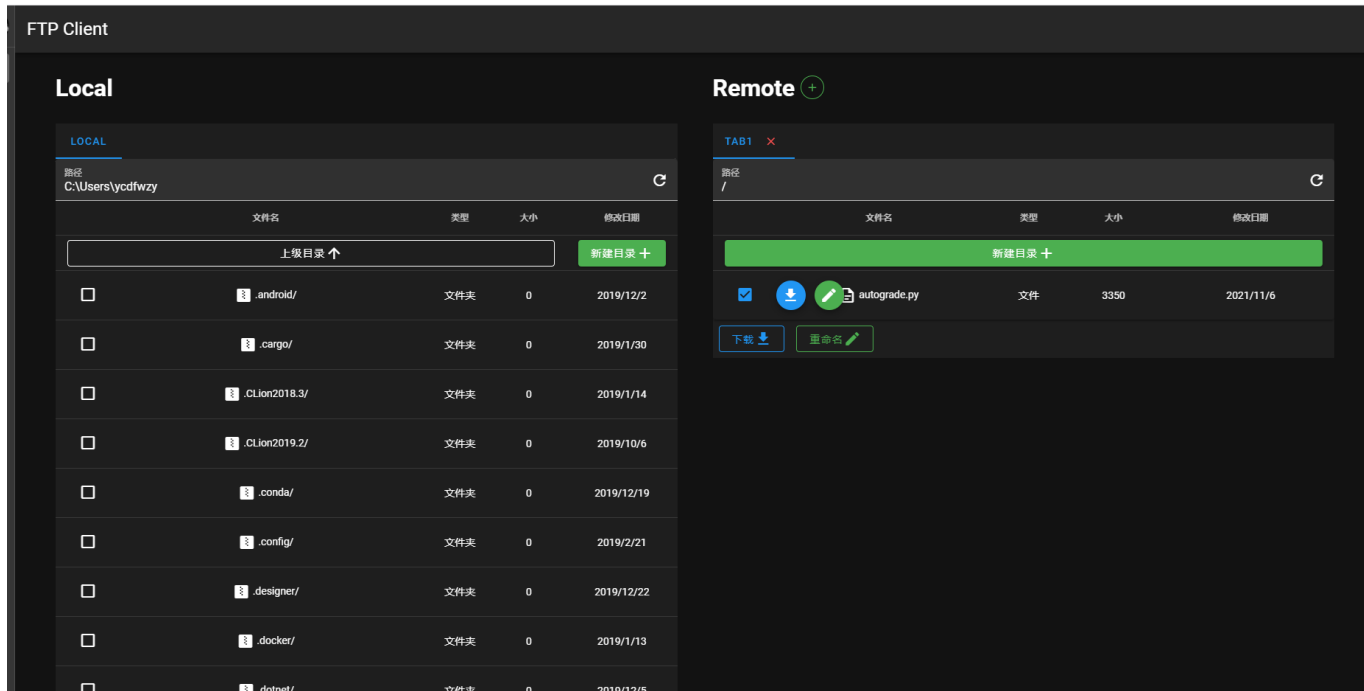
#### 4. 实验感想

这次从零开始根据协议实现 FTP server 是一个非常大的挑战, 从最开始的无从下手到中间的各种 Bug 都几度让人心态炸裂。比较好的是从布置开始就一点一点写, 战线拉得很长, 才能在DDL前一天就写完了功能改好了致命的 Bug。

编写 Client 的时候设想了很多很好的功能, 但是在实现的中途因为各种函数设计的不合理导致了严重的 bug, 改着改着时间就不够了, 不得不放弃了一些原有的想法, 最后只实现了很简陋的客户端, 十分遗憾。

# Previous Outstanding Project

## Student B



# Previous Outstanding Project

Student C

- **Handle the following exceptions:**
  - Catch password format exception;
  - Catch PORT data socket creating exception;
  - Catch PASV listening exception;
  - Catch RETR STOR I/O exception;
  - Catch thread creating exception;
  - Catch directory management exception;
  - Catch file management exception;
  - Catch illegal file path exception;
  - Catch control socket exception;
  - Catch data socket exception;

Good Luck!