Introduction to C Programming

Lecture 9: head files

Wenjin Wang
wangwj3@sustech.edu.cn

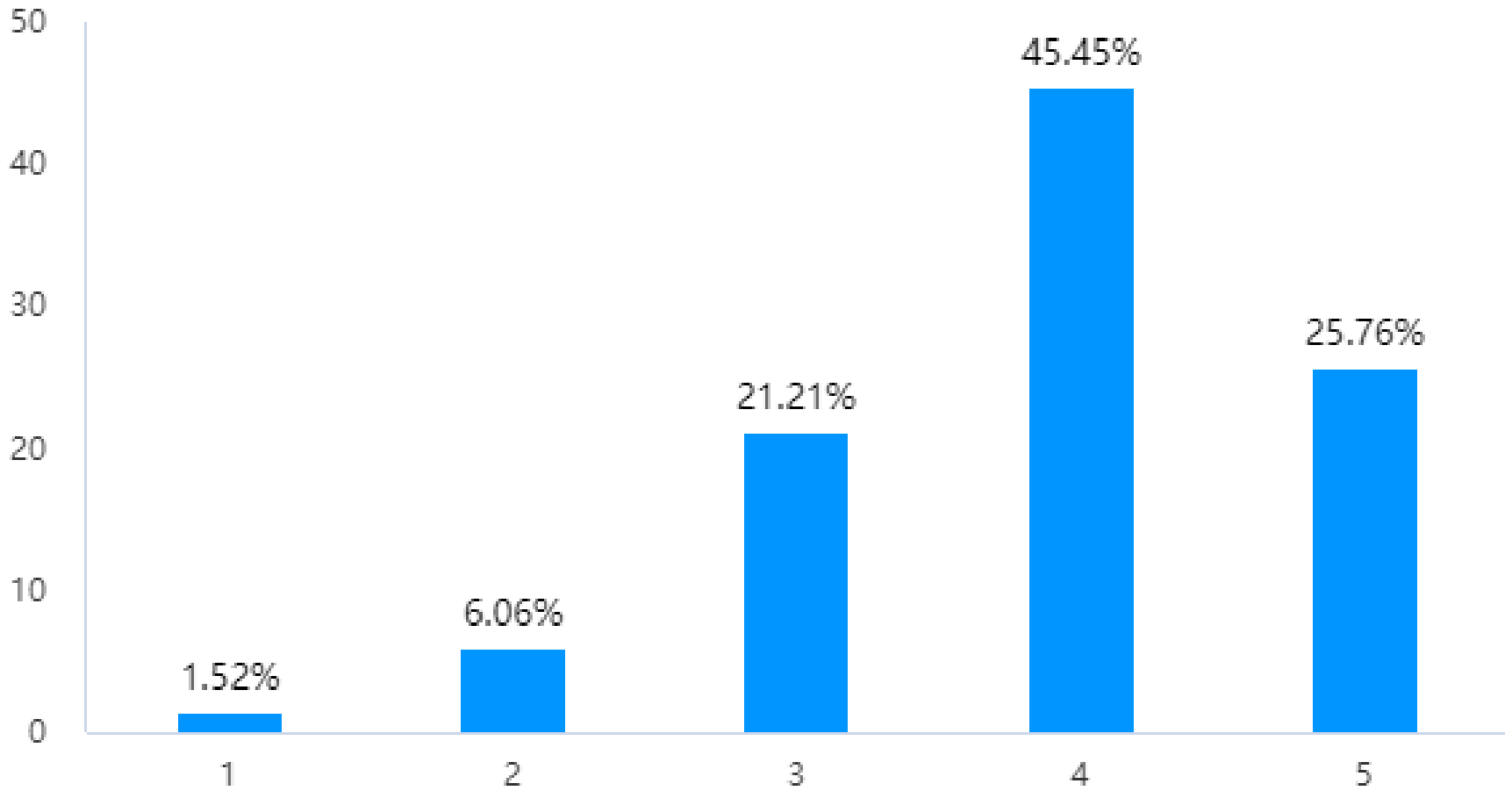
11-4-2022

课程调研

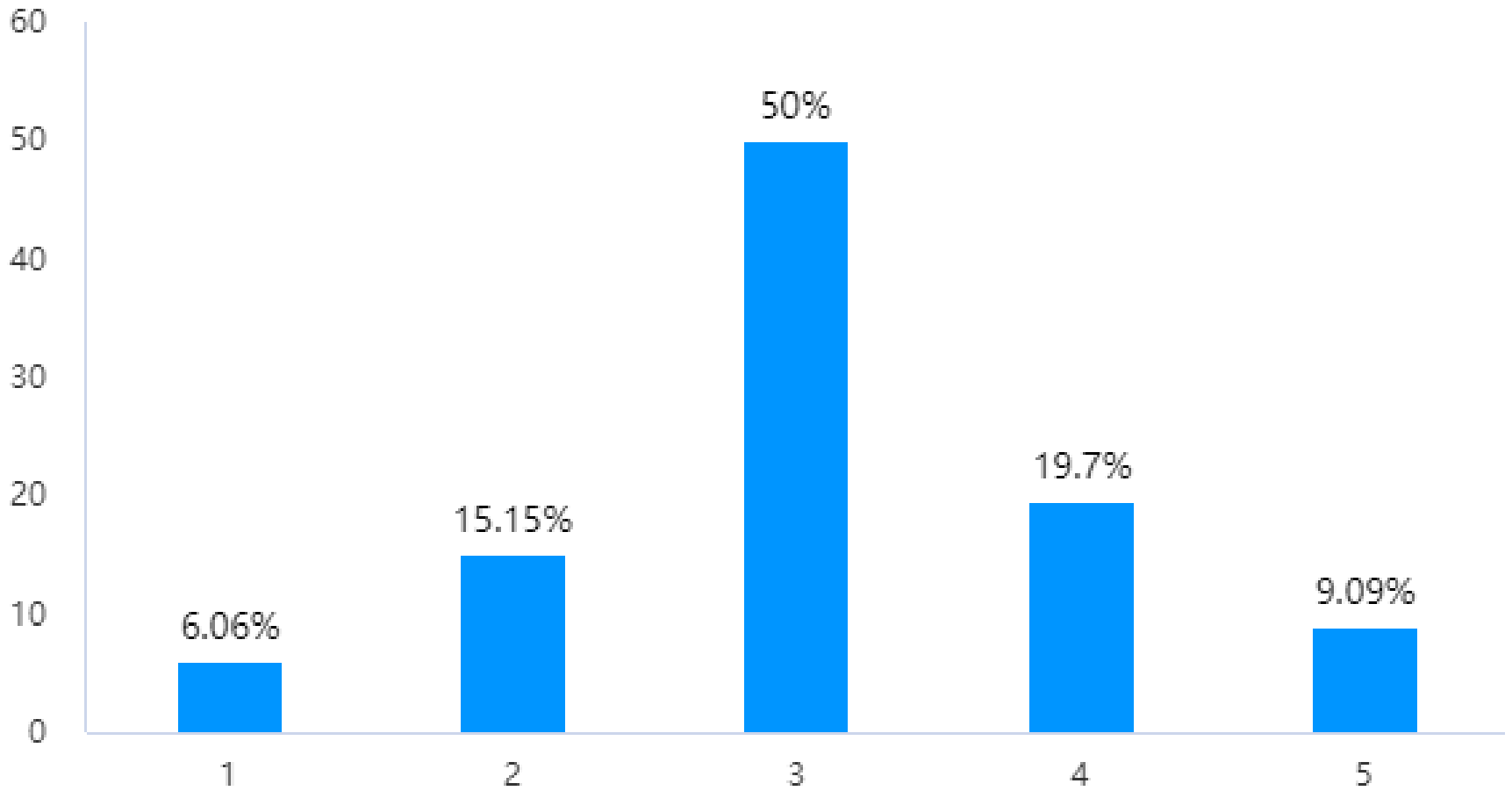
66人参加调研（共87人）

75.9%

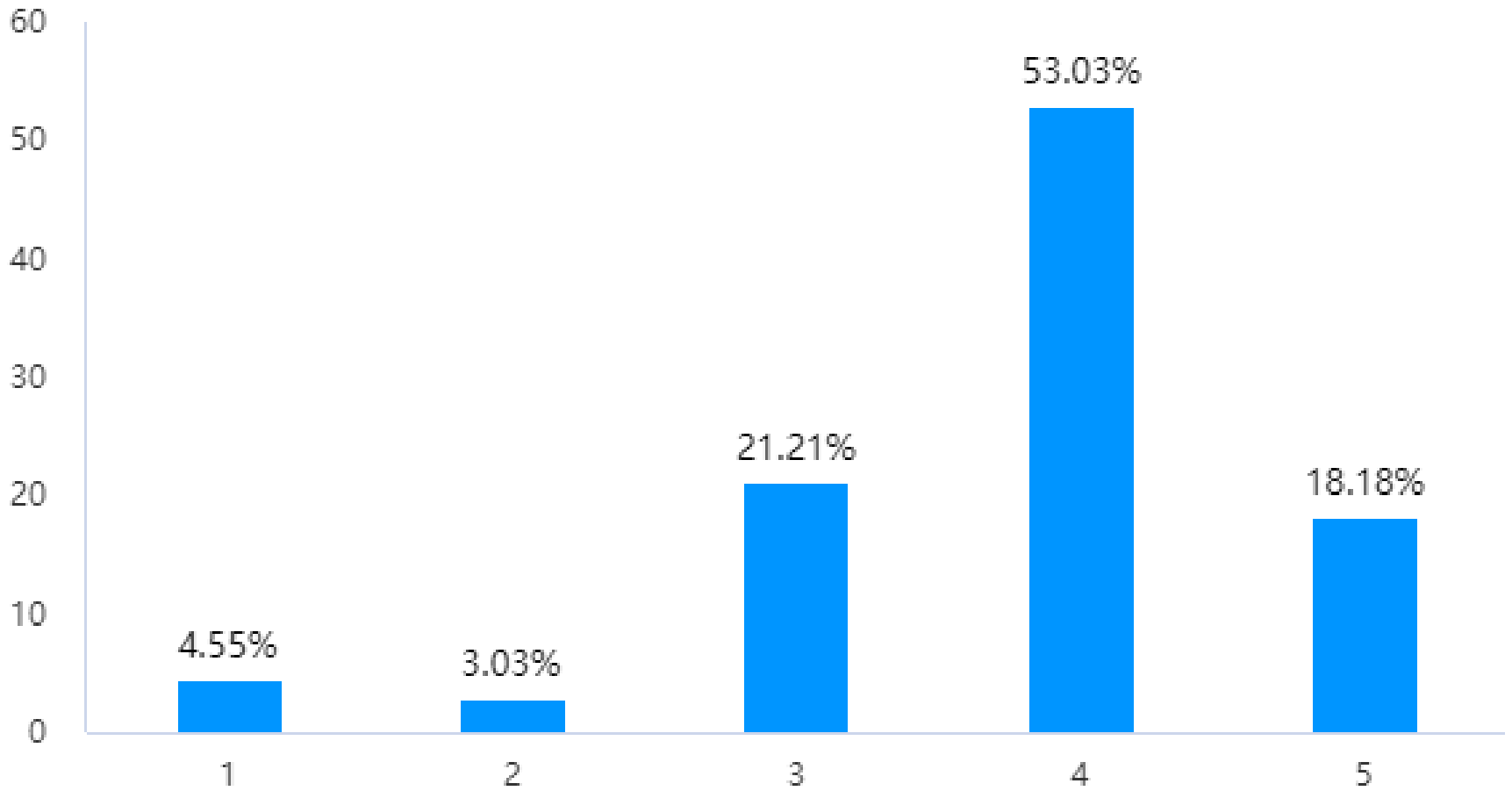
课程整体难度（1 非常简单 5 非常困难）



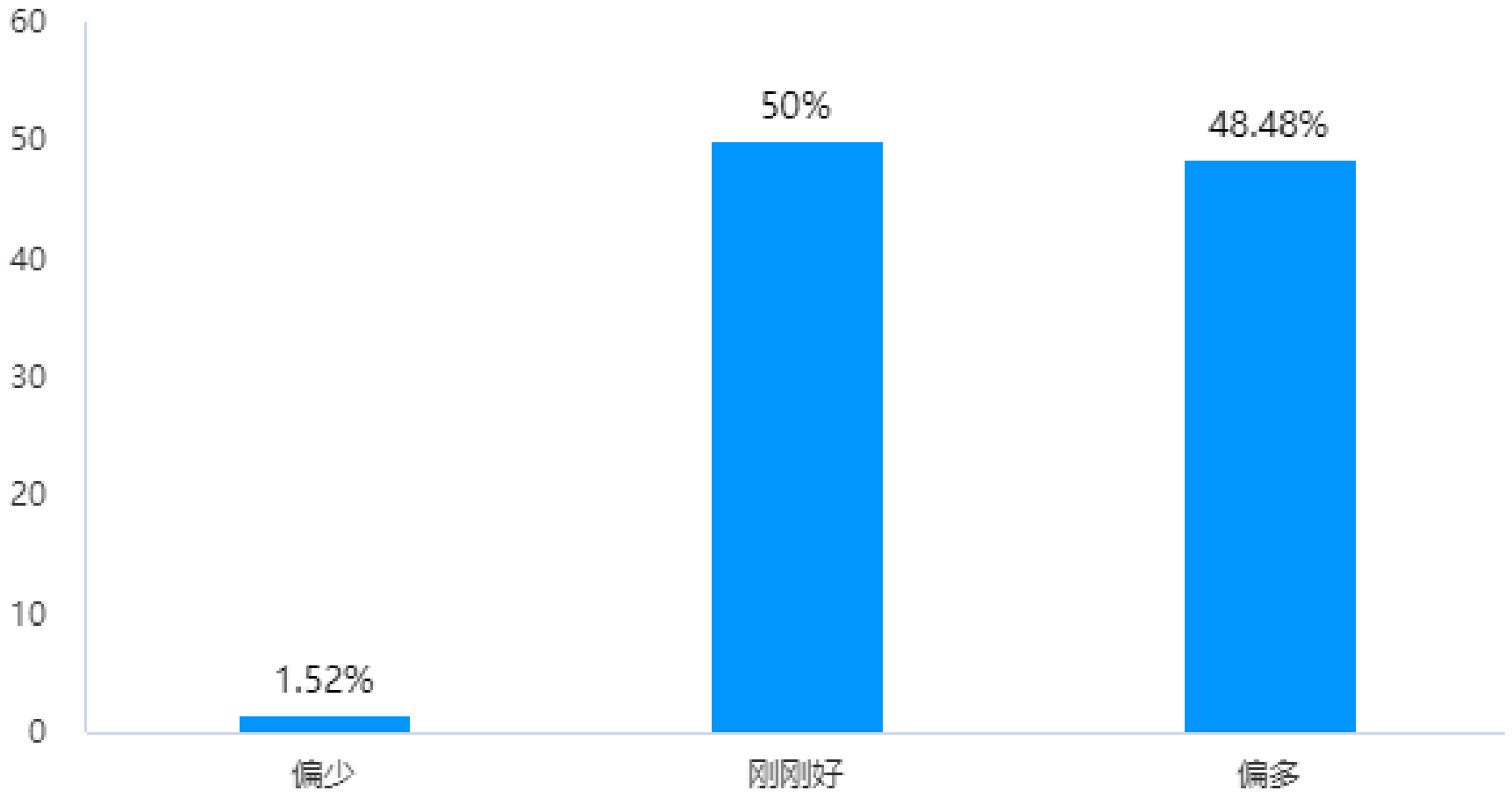
对现有知识点的掌握程度（1 非常好 5 非常差）



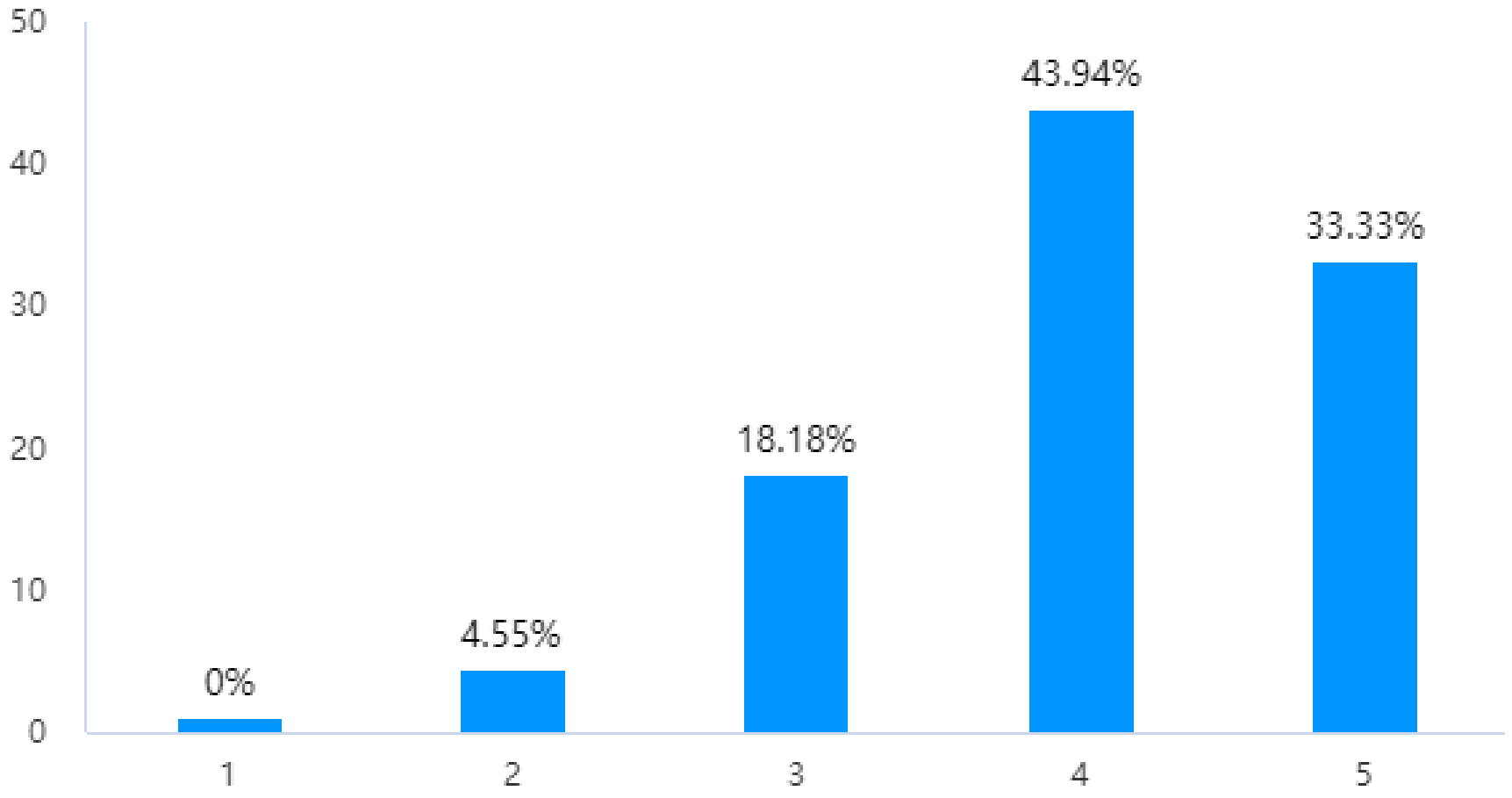
课程作业难度（1 非常简单 5 非常困难）



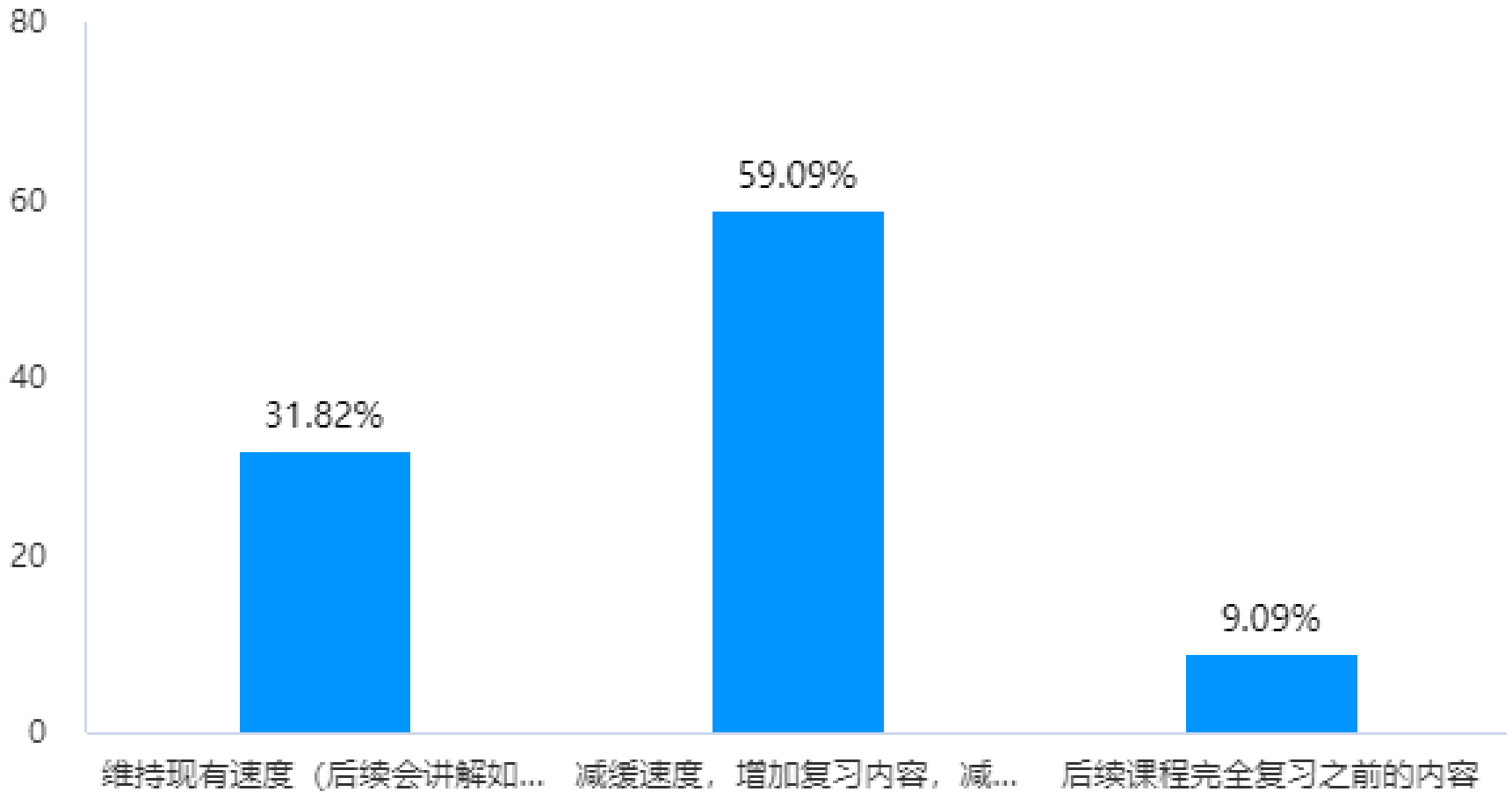
你觉得现在作业量怎么样



课程进度（1 非常慢 5 非常快）



你希望接下来课程进度的安排



反馈意见

例子

- 多重实际问题，重应用
- 希望可以添加程序没有逻辑错误、编译器不报错却无法运行的情况的讲解

习题

- 可以花大概五分钟的时间粗略讲解一下上节课的作业，便于巩固理解
- 能不能抽一两节课专门讲讲习题
- 例子给的再丰富全面一点，适当覆盖作业

进度

- 快讲讲机器学习板块(doge)期待期待
- 真的太快了，我觉得之后的八节课可以完全用来复习

改善举措

- 增加复习并讲解习题（拟3次课）
- 适当调整讲解的速度和课程进度
- 两周无课期间设置答疑时间（2小时），不想躺的可来

Course syllabus

Nr.	Lecture	Date
1	Introduction	2022.9.9
2	Basics	2022.9.16
3	Decision and looping	2022.9.23
4	Array & string	2022.9.30
5	Functions	2022.10.9 (补)
6	Pointer	2022.10.14
7	Self-defined types	2022.10.21
8	I/O	2022.10.28

Nr.	Lecture	Date
9	Head files	2022.11.4
10	Review of lectures I	2022.11.25
11	Review of lectures II	2022.12.2
12	Review of lectures III	2022.12.9
13	AI in C programming	2022.12.16
14	AI in C programming	2022.12.23
15	AI in C programming	2022.12.30
16	Summary	2023.1.6

Recap last lecture

- Three types of I/O: **user I/O**, **file I/O** and **socket I/O**.
- **User I/O**: read/write single char (getchar, putchar), read/write a group of chars (gets, puts), formatted reading/writing (scanf, printf).
- **File I/O**: ASCII file (.txt, .csv), binary file (.bin). Four basic file operations: open file, close file, write file, read file.
- **Socket I/O**: client and server. Three essential parts: IP address, port number, transmission protocol (TCP, UDP).
- You can create a socket connection to transfer file data?!

Objective of this lecture

Review I/O (file & socket) and you can work with head files!

Content

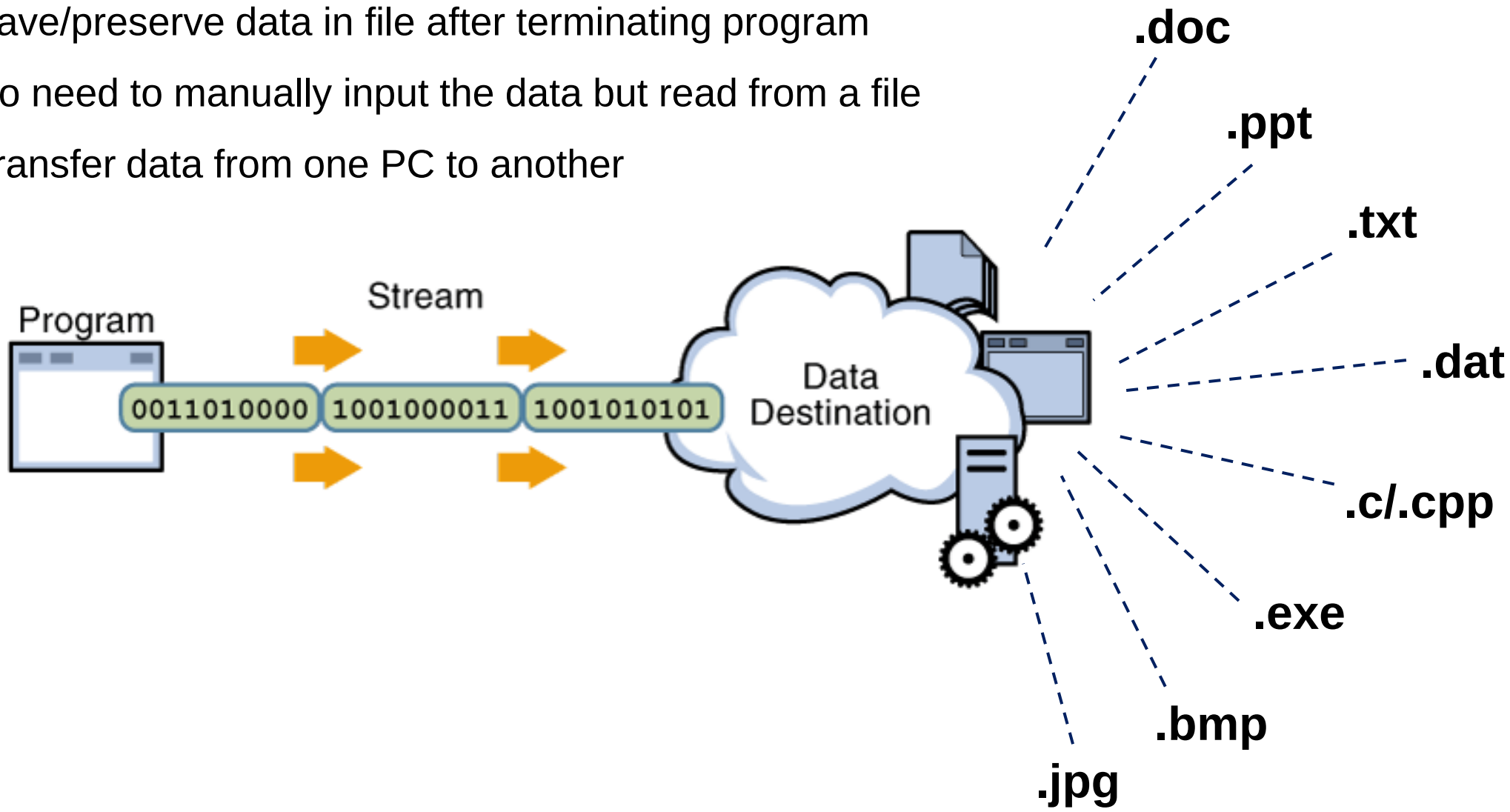
1. File I/O & socket I/O
2. Head files
3. Multi-threading (不做要求！)

Content

1. File I/O & socket I/O
2. Head files
3. Multi-threading (不做要求！)

File I/O

- Save/preserve data in file after terminating program
- No need to manually input the data but read from a file
- Transfer data from one PC to another



File formats

ASCII file

binary file

.txt

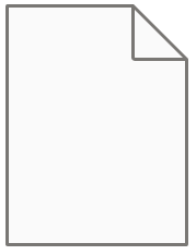
Plain **text**
(data in characters)

.csv

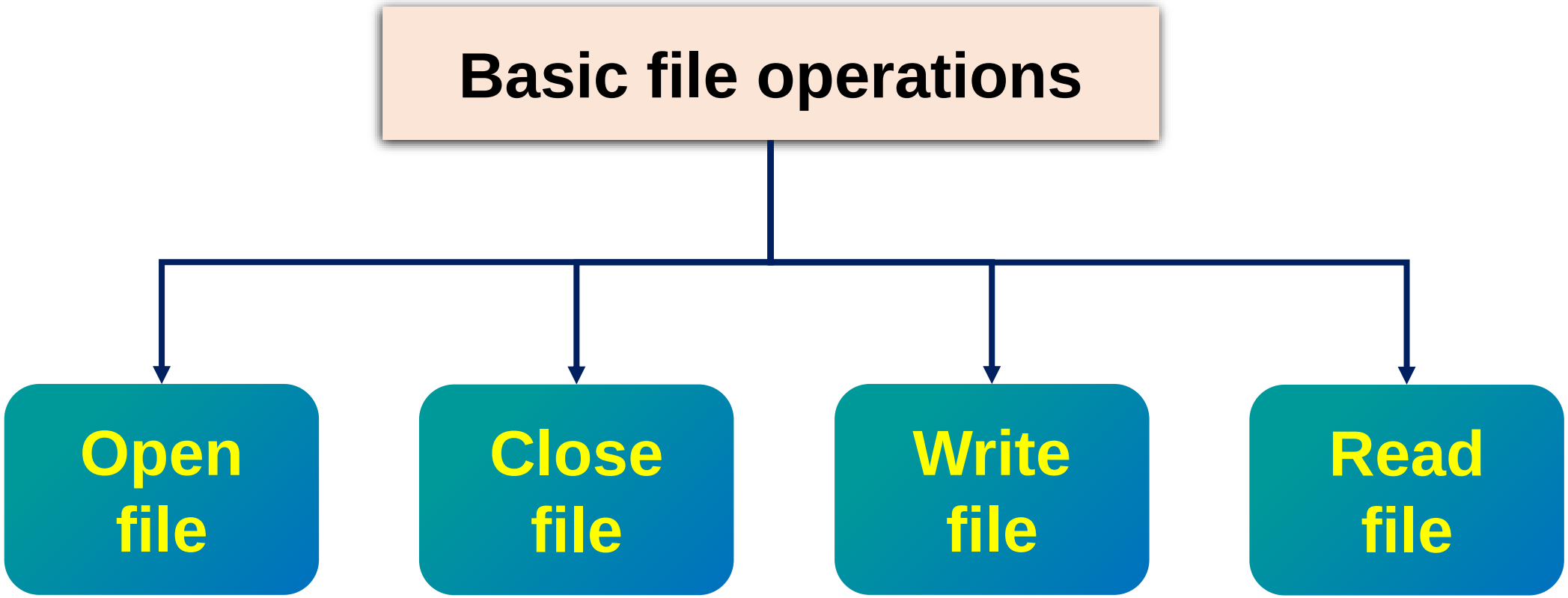
Comma **S**eparated **V**alues
(data structured by “,”)

.bin

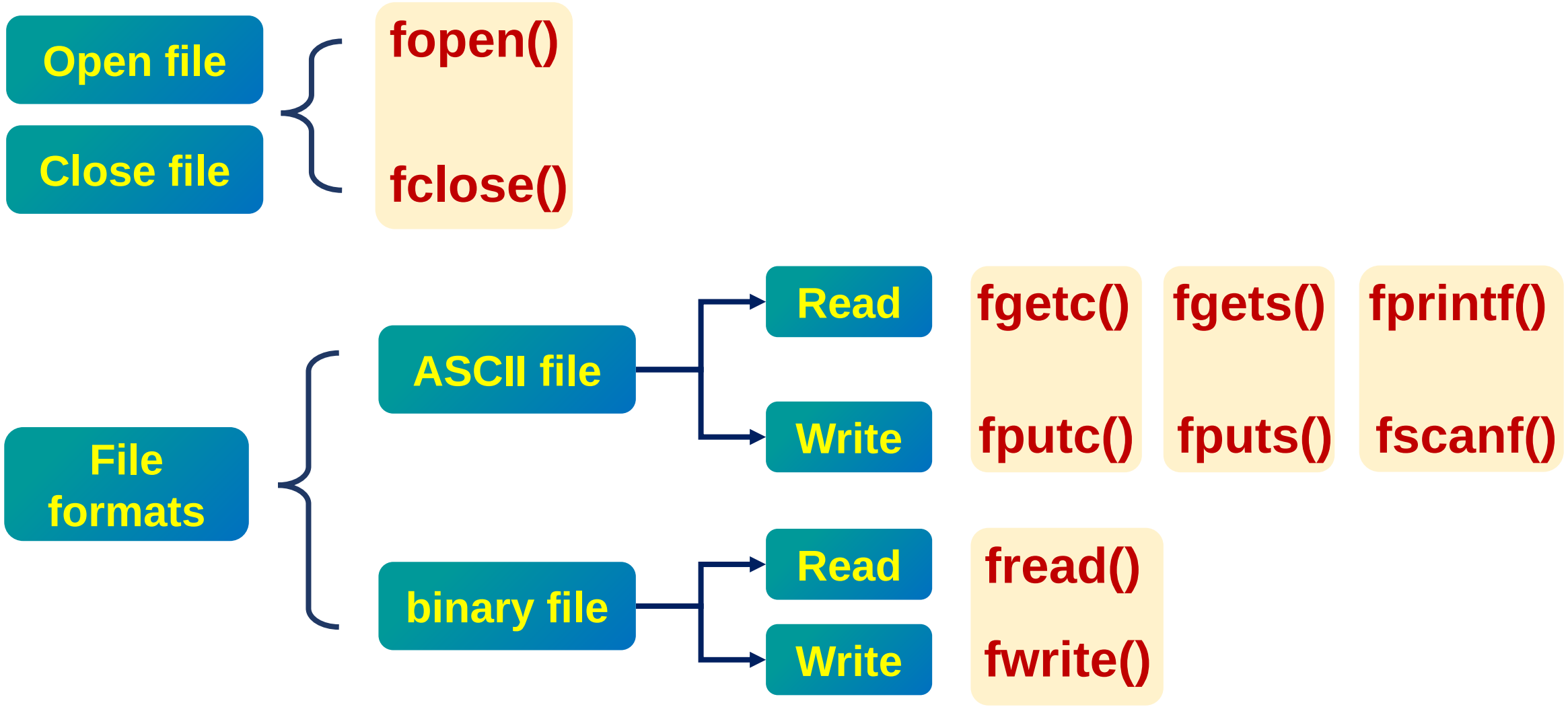
Binary values
(data in 0 and 1)



File I/O functions



File I/O functions



Open file

Declare a FILE pointer (FILE belongs to #include<stdio.h>)

```
FILE* fp;

fp = fopen(const char* filename, const char* mode);
```

Path of the file in the system

Absolute path
（绝对路径）

Relative path
（相对路径）

Mode (模式)

read write append

Open file: how to define path?

File name

File type

```
"data.txt"
```

```
"C:\\Users\\wenji\\Desktop\\c files\\data.txt"
```

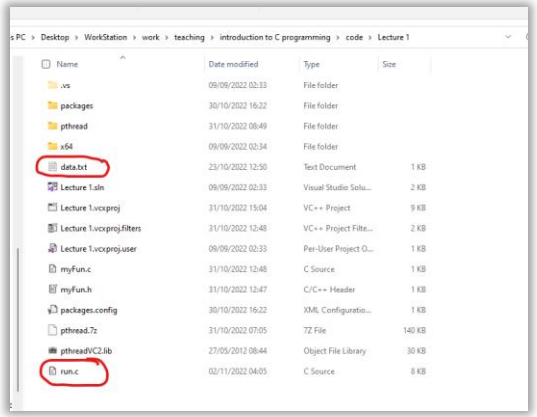
File path

Open file: how to define path?

```
FILE* fp;
```

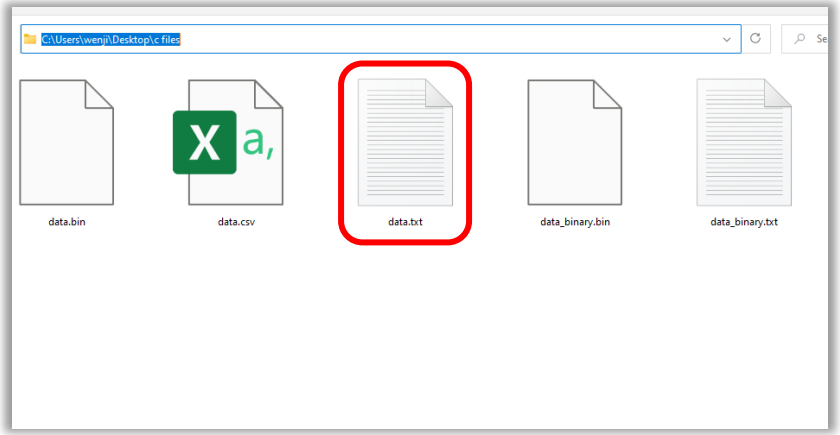
当前.c文件的路径（相对路径）

```
fp = fopen("test.txt", "w");
```




```
fp = fopen("C:\\Users\\wenji\\Desktop\\c
files\\data.txt", "w");
```

系统的绝对路径



Open file

有些编辑器会报错！

```
error C4996: 'fopen': This function or variable may be unsafe.
Consider using fopen_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details.
```

解决方案1：在最前面添加 #include _CRT_SECURE_NO_WARNINGS

解决方案2：使用fopen_s函数

Open file: how to define mode?

r = read | w = write | a = append | b = binary

To open ASCII files (.txt, .csv):

"r", "w", "a", "r+", "w+", "a+" → 混合型

To open binary file (.bin):

"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

Close file

Input the FILE pointer

```
FILE *fp;
```

```
int fclose(FILE * fp);
```

- ✓ Flushes data pending in the buffer to file
- ✓ Closes the file
- ✓ Releases memory used for the file

Open and close a file

```
#include <stdio.h>
main()
{
    FILE* fp;

    fp = fopen("test.txt", "w+");

    // …

    fclose(fp);
}
```

Can be .txt, .bin, .csv

In a pair

Example of opening files

bin file

```c
#include <stdio.h>

main()
{
    FILE* fp;
    fp = fopen("test.bin", "rb");
    fp = fopen("test.bin", "wb");
    fp = fopen("test.bin", "ab");

    fp = fopen("test.bin", "r+b");
    fp = fopen("test.bin", "w+b");
    fp = fopen("test.bin", "a+b");
}
```

csv file

```c
#include <stdio.h>

main()
{
    FILE* fp;
    fp = fopen("test.csv", "r");
    fp = fopen("test.csv", "w");
    fp = fopen("test.csv", "a");

    fp = fopen("test.csv", "r+");
    fp = fopen("test.csv", "w+");
    fp = fopen("test.csv", "a+");
}
```

Write\read a file

Write/read a single character:

```
int fputc(int c, FILE *fp);
int fgetc(FILE *fp);
```

Write/read a group of characters:

```
fputs(const char *s, FILE *fp);
fgets(char *buf, int n, FILE *fp);
```

Write/read formatted characters:

```
fprintf(FILE *fp, const char *format, ...);
fscanf(FILE *fp, const char *format, ...);
```

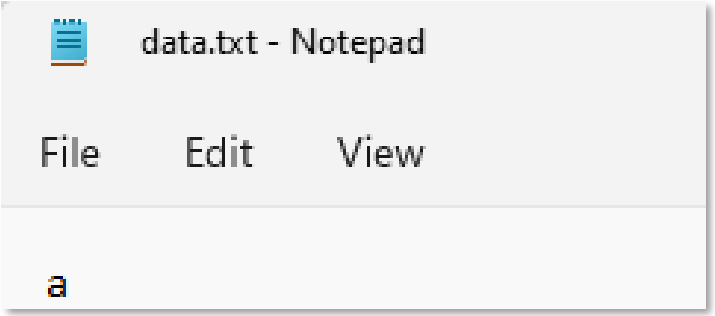
Write/read binary file:

```
fwrite(const void *ptr, int size_of_elements, int number_of_elements, FILE *fp);
fread(void *ptr, int size_of_elements, int number_of_elements, FILE *fp);
```

Write/read single character

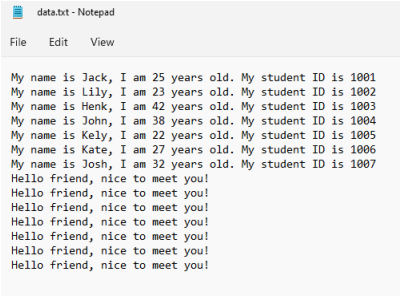
Write

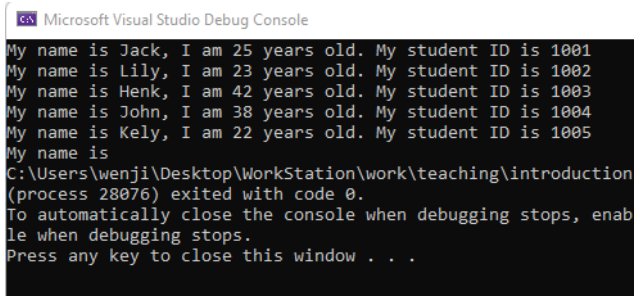
```c
#include <stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "w+");
    char data = 'a';
    fputc(data, fptr);

    fclose(fptr);
}
```



Read

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    for (int i = 0; i < 300; i++)
    {
        char c = fgetc(fptr);
        printf("%c", c);
    }
    fclose(fptr);
}
```

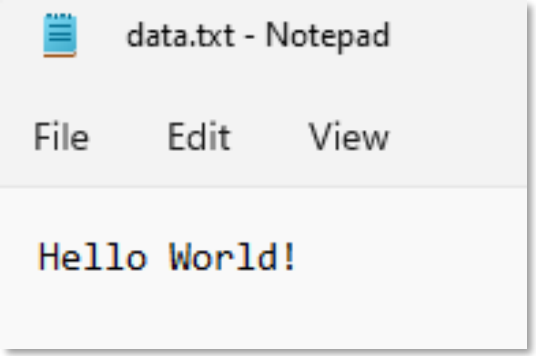




Write/read a group of characters

Write

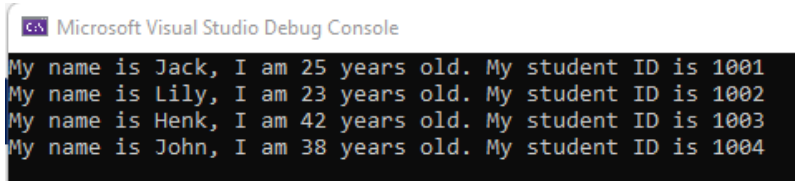
```c
#include<stdio.h>
main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "w");

    char data[] = "Hello World!";
    fputs(data, fptr);

    fclose(fptr);

}
```



Read

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char data[300];

    fgets(data, 100, fptr); printf("%s", data);

    fgets(data, 100, fptr); printf("%s", data);

    fgets(data, 100, fptr); printf("%s", data);

    fgets(data, 100, fptr); printf("%s", data);

    fclose(fptr);
}
```

Length of string (N-1, last is null)



Write/read formatted characters

Write

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "w");

    char format[] = "My name is %s, I am %d
years old. My student ID is %d\n";

    fprintf(fptr, format, "Jack", 25, 1001);
    fprintf(fptr, format, "Lily", 23, 1002);
    fprintf(fptr, format, "Henk", 42, 1003);
    fprintf(fptr, format, "John", 38, 1004);
    fprintf(fptr, format, "Kely", 22, 1005);
    fprintf(fptr, format, "Kate", 27, 1006);
    fprintf(fptr, format, "Josh", 32, 1007);

    fclose(fptr);
}
```

Writing mode

data.txt - Notepad

File Edit View

```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
```

Read

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char str[20];
    fscanf(fptr, "%s", str);

    printf("%s", str);

    fclose(fptr);
}
```

Microsoft Visual Studio Debu

```
My
C:\Users\wenji\Desktop\
(process 6728) exited w
To automatically close
le when debugging stops
Press any key to close
```

Read single word

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char str1[20], str2[20];
    fscanf(fptr, "%s %s", str1, str2);

    printf("%s, %s", str1, str2);

    fclose(fptr);
}
```

Microsoft Visual Studio Deb

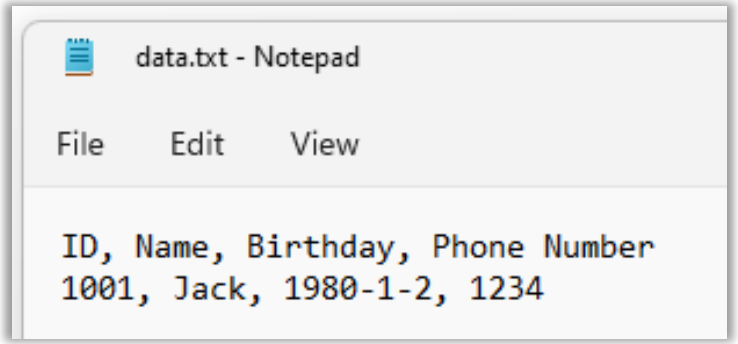
```
Hello, my
C:\Users\wenji\Desktop
(process 31468) exited
To automatically close
le when debugging stop
Press any key to close
```

Read 2 words

Write/read formatted characters

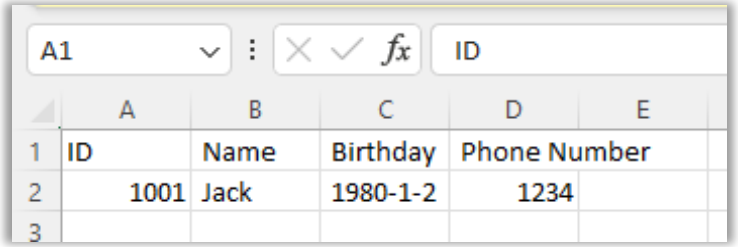
txt format

```c
#include<stdio.h>
main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "w");
    fprintf(fptr, "ID, Name, Birthday, Phone Number\n");
    fprintf(fptr, "%d, %s, %s, %d\n", 1001, "Jack", "1980-1-2", 1234);
    fclose(fptr);
}
```



csv format

```c
#include<stdio.h>
main()
{
    FILE* fptr;
    fptr = fopen("data.csv", "w");
    fprintf(fptr, "ID, Name, Birthday, Phone Number\n");
    fprintf(fptr, "%d, %s, %s, %d\n", 1001, "Jack", "1980-1-2", 1234);
    fclose(fptr);
}
```



Write/read binary file

Write

```c
#include<stdio.h>

typedef struct student {
    char name[100];
    int id;
    float average;
}stud;

main()
{
    FILE* fptr;
    fptr = fopen("data.bin", "wb");

    stud data[] = { {"Kate", 1001, 90},
{"Jack", 1002, 94}, {"Mike", 1003, 85} };
    fwrite(data, sizeof(data), 1, fptr);

    fclose(fptr);
}
```

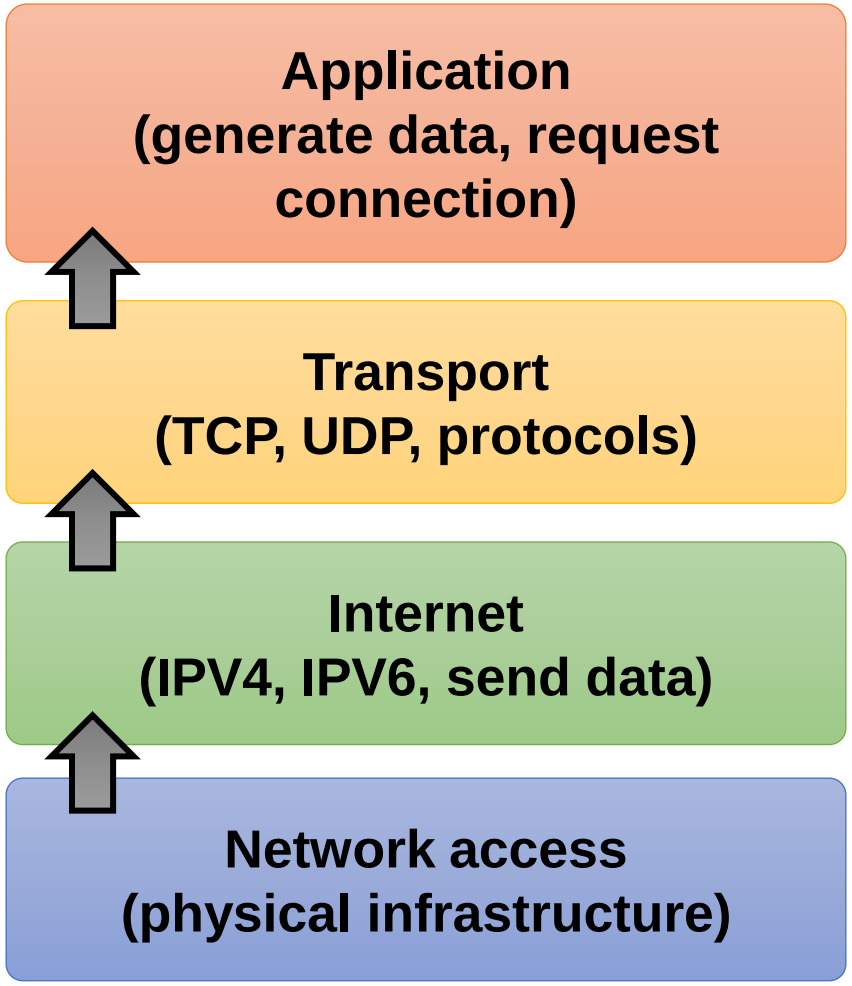
Read

```c
#include<stdio.h>

typedef struct student {
    char name[100];
    int id;
    float average;
}stud;

main()
{
    FILE* fptr;
    fptr = fopen("data.bin", "rb");

    stud data_read[3];
    fread(&data_read, sizeof(data_read), 1, fptr);
    printf("%s %d %f\n", data_read[0].name,
data_read[0].id, data_read[0].average);
    printf("%s %d %f\n", data_read[1].name,
data_read[1].id, data_read[1].average);
    printf("%s %d %f\n", data_read[2].name,
data_read[2].id, data_read[2].average);

    fclose(fptr);
}
```

Internet I/O in life

QQ Wechat Tiktok

4-layers model



What is socket?

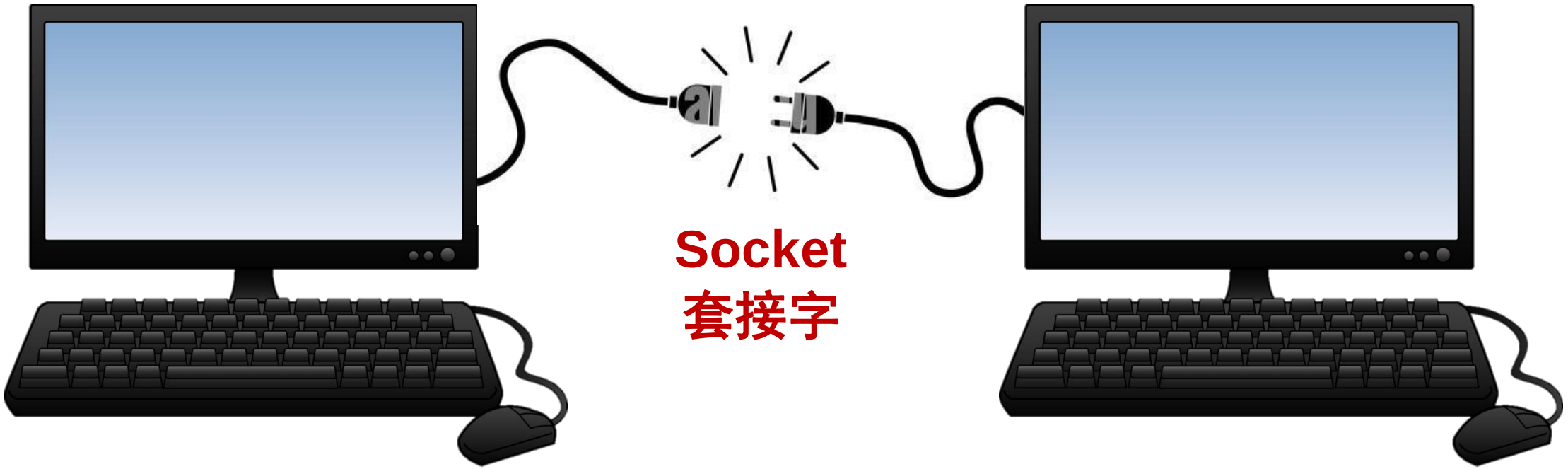
套接字

套接字（socket）是通信的基石，是网络通信过程中端点的抽象表示，包含三种必要信息：**通信协议(TCP/UDP)**，**IP地址**，**端口**。

What is socket?

IP address	Port
10.0.0.201	1234

IP address	Port
10.0.0.101	101



What is socket?

IP address	Port
10.0.0.201	1234

Create socket → Connect → Send → Close socket

1. Socket socket = createSocket(**type = "TCP")**
2. connect(socket, **address = "1.2.3.4"**, **port = "80"**)
3. send(socket, "Hello, world!")
4. close(socket)

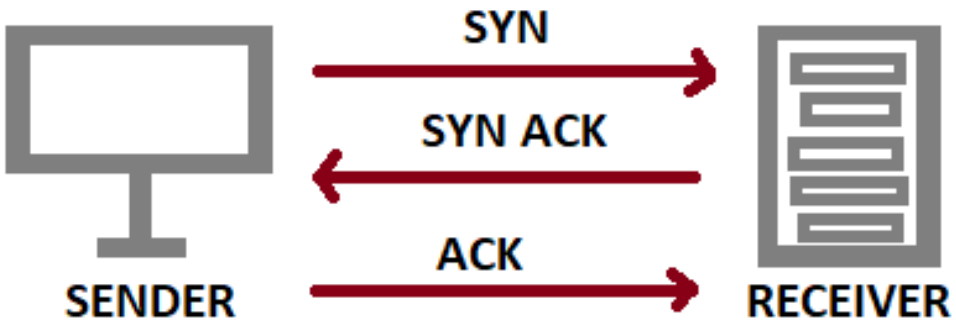
What is socket?

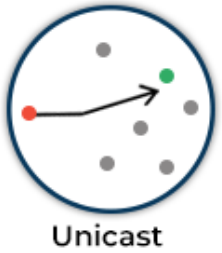
Use socket to create an internet connection, rest are file I/O!!!

- Commination protocol
- IP address
- Porter

Communication protocol

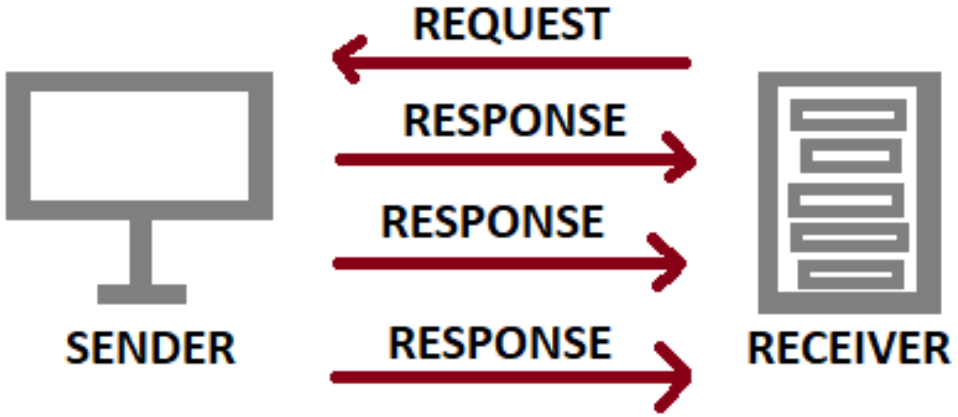
Transmission Control Protocol (TCP)

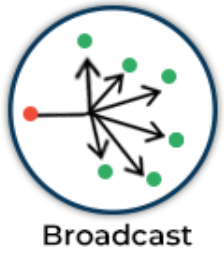




- Connection-oriented, 1-to-1
- Slower but reliable transfer
- Source intensive
- Typical applications: emails, web browsing, file transfer, etc.

User Datagram Protocol (UDP)



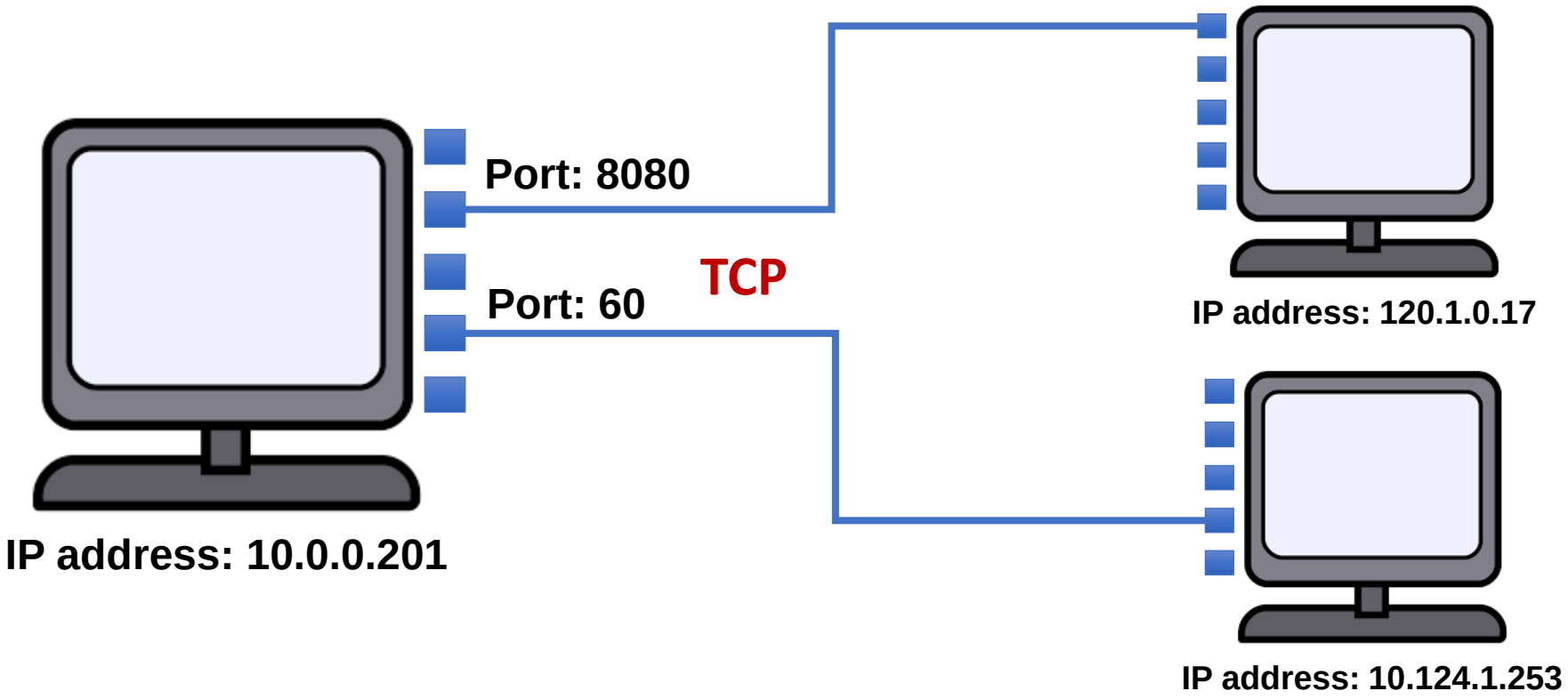


- Message-oriented, 1-to-N
- Faster but no guaranteed transfer
- Lightweight
- Typical applications: live streaming, online games, etc.

<u>UDP isn't that bad in reality</u>

IP address & port number

Socket address		IP address		Port number
10.0.0.201:8080	=	10.0.0.201	+	8080



IP address & port number

IP address	Port number
10.0.0.201	8080

It identifies a machine in the network, must be **unique**!

It identifies a particular **application or process** in a system.

IPV4

192.168.5.18

4 octave x 8 bit = 32 bits（十进制）

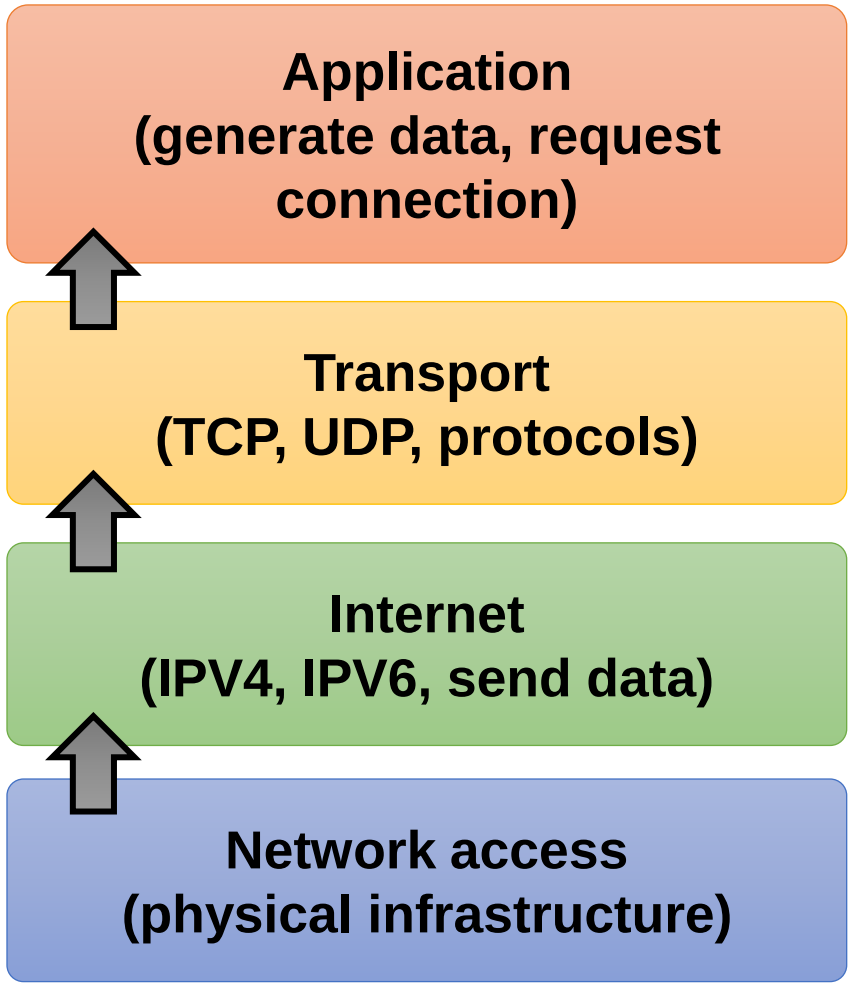
IPV6

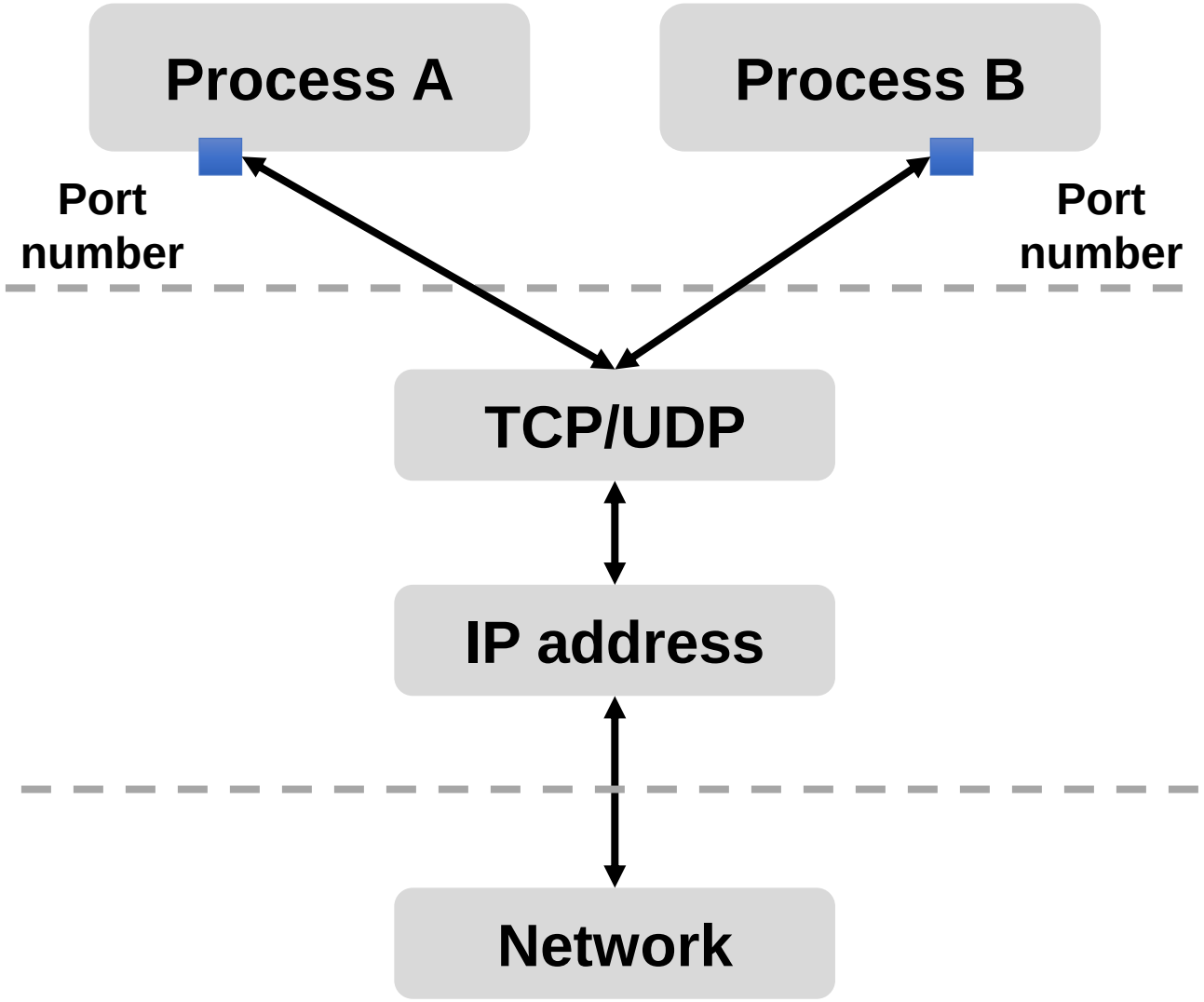
50b2:6400:0000:0000:6c3a:b17d:0000:10a9

8 octave x 16 bit = 128 bits（十六进制）

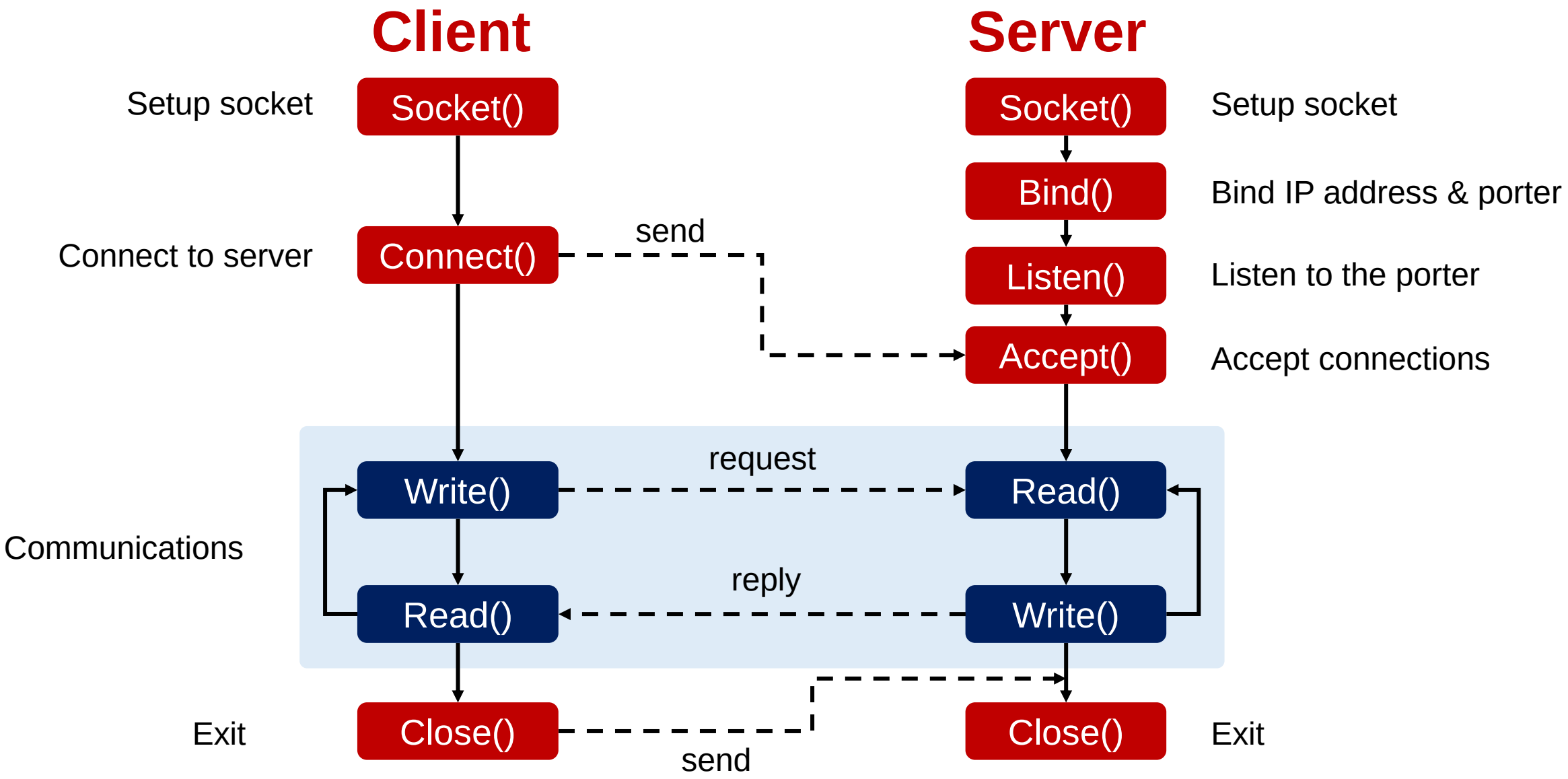
Overview network model

4-layers model





Overview of a socket



socket()

int socket(int domain, int type, int protocol);

Socket标识

协议域

AF_INET6: IPV6

AF_INET: IPV4

AF_LOCAL: absolute path

类型

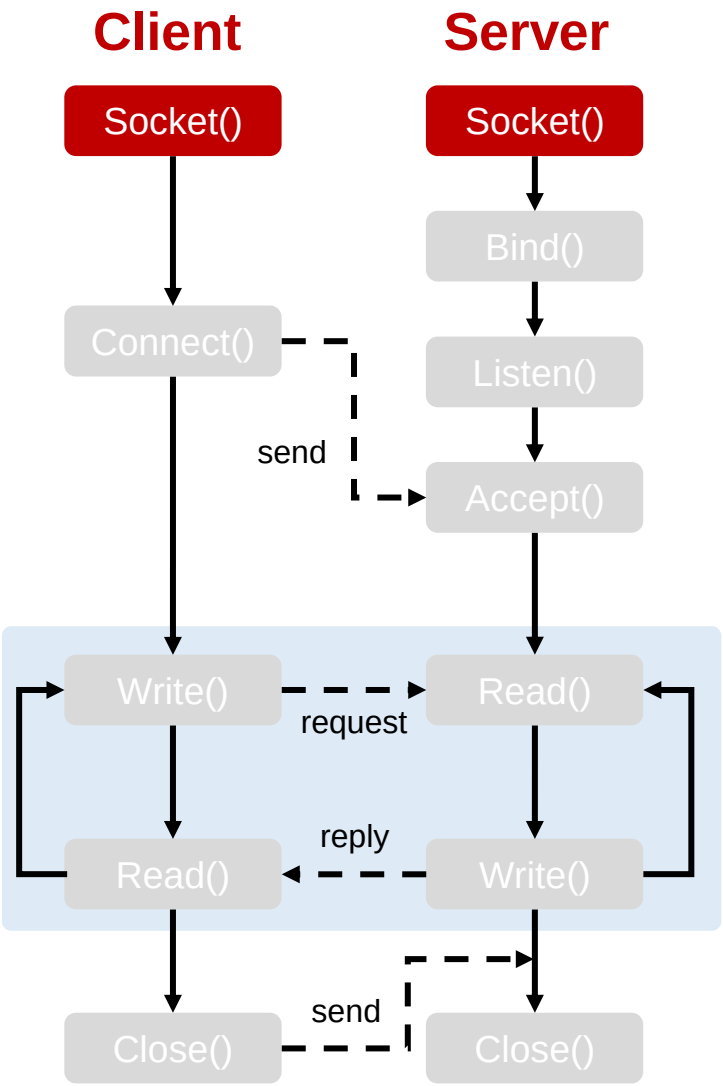
SOCK_STREAM: Stream Sockets (流格式套接字)

SOCK_DGRAM: Datagram Sockets (数据报格式套接字)

协议

IPPROTO_TCP: TCP传输协议

IPPTOTO_UDP: UDP传输协议



socket()

int socket(int domain, int type, int protocol);

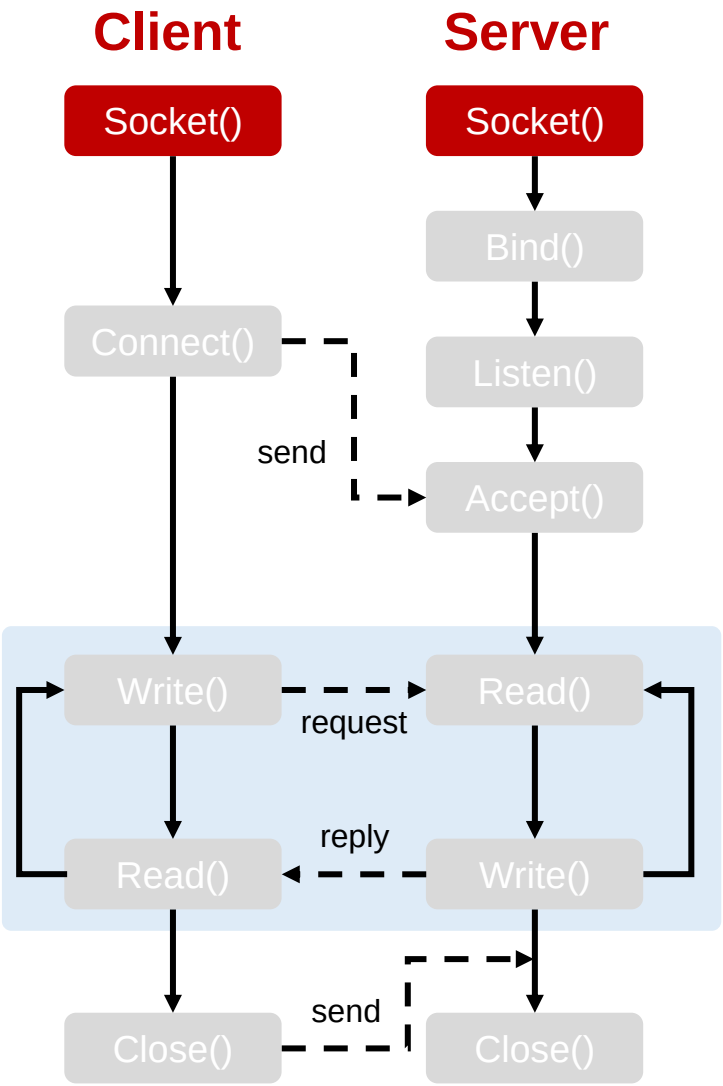
协议域

AF_INET6: IPV6

AF_INET: IPV4

AF_LOCAL: absolute path

```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::701a:d780:be90:c147%19
   IPv4 地址 . . . . . . . . . . . . : 192.168.3.75
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.3.1
```



socket()

int socket(int domain, int type, int protocol);

类型

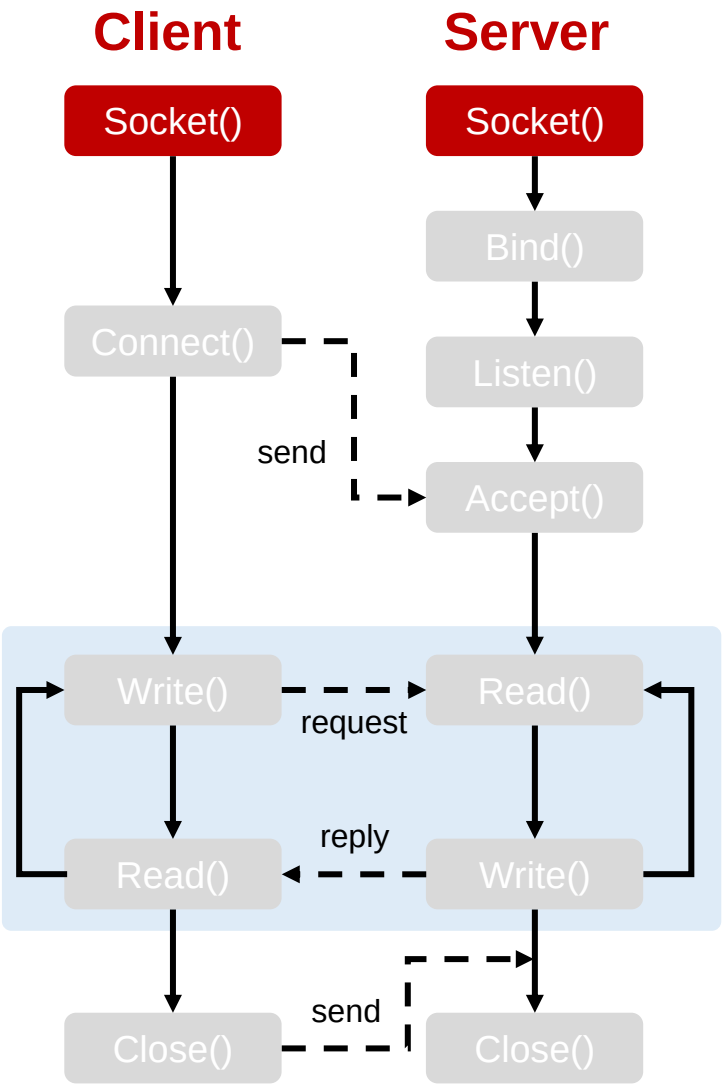
SOCK_STREAM: Stream Sockets
(流格式套接字)

高质量：SOCK_STREAM是一种可靠的、双向的通信数据流，数据可以准确无误地到达另一台PC，如果损坏或丢失，可以重新发送。

SOCK_DGRAM: Datagram Sockets
(数据报格式套接字)

高效率：SOCK_DGRAM只关心传输速度，不作数据校验。如果数据丢失或损坏，无法重新传输。

腾讯会议
Tencent Meeting



socket()

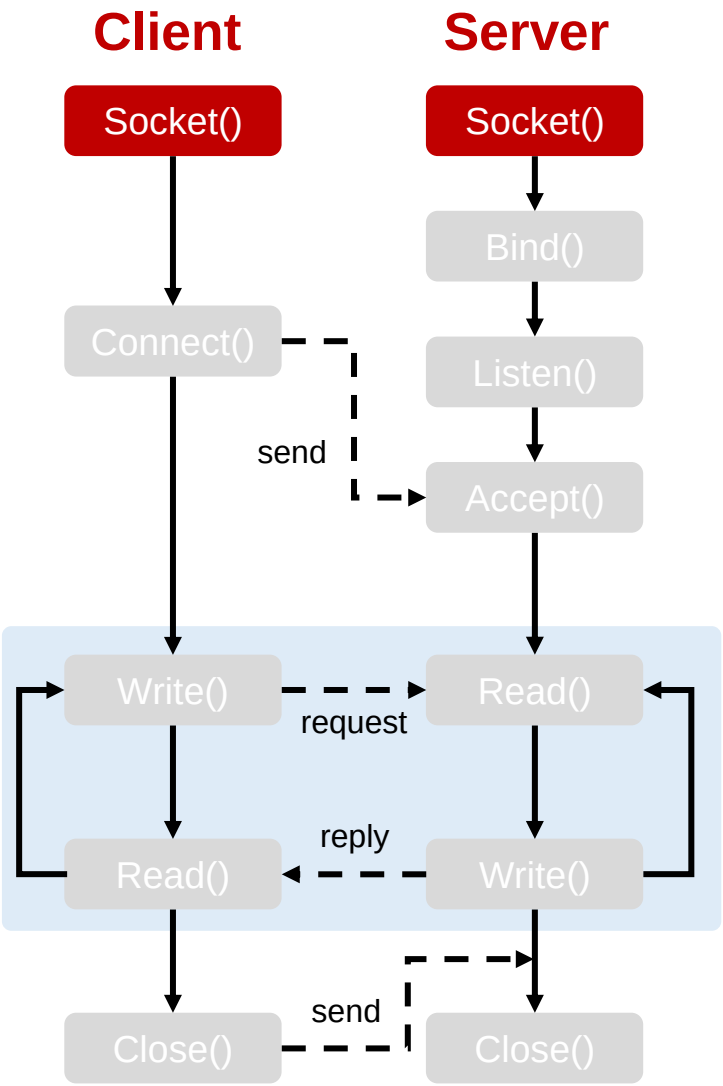
int socket(int domain, int type, int protocol);

协议

IPPROTO_TCP: TCP传输协议 (Transmission Control Protocol) ⟷ **SOCK_STREAM**

Protocol must match with type

IPPTOTO_UDP: UDP传输协议 (User Datagram Protocol) ⟷ **SOCK_DGRAM**



socket()

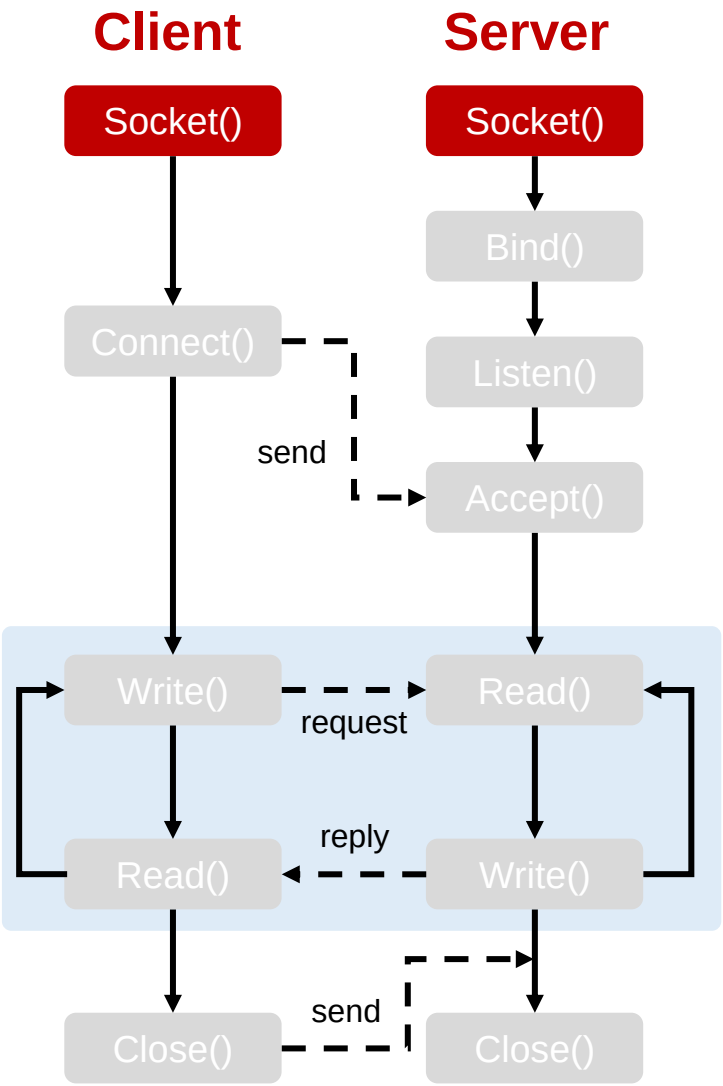
int socket(int domain, int type, int protocol);

IPV4 + STREAM + TCP (high-quality communication)

```
SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

IPV4 + DGRAM + UCP (high-speed communication)

```
SOCKET sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```



bind()

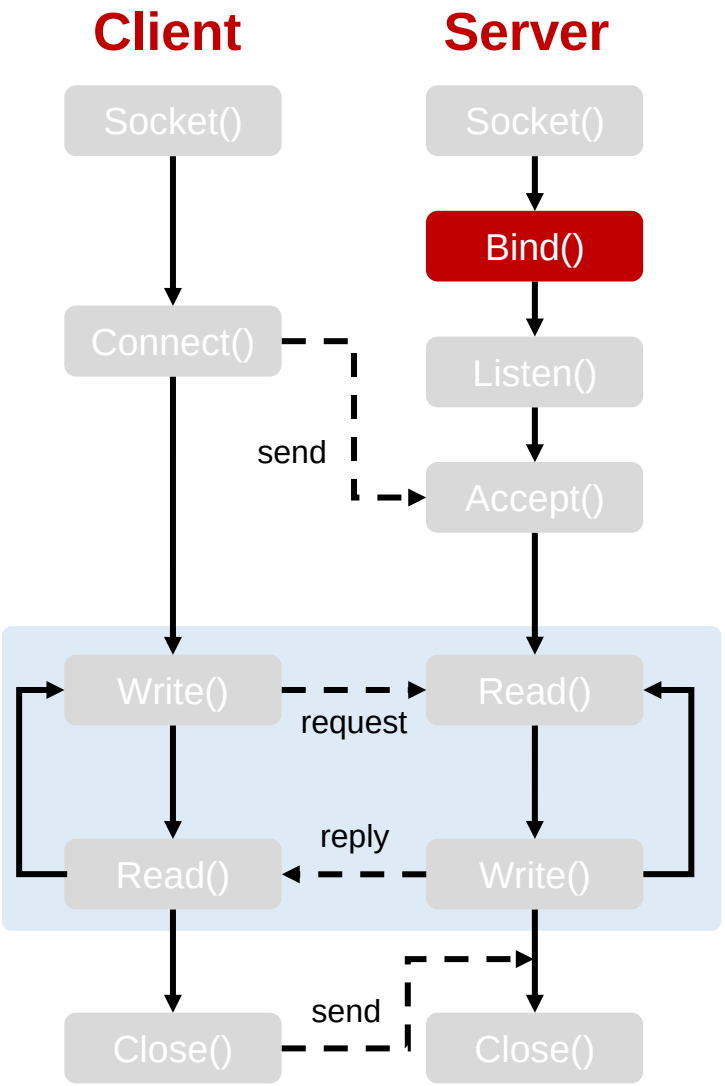
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Socket标识

协议地址长度

协议地址
（IP地址，端口号）

```
struct sockaddr_in sin;
sin.sin_family = AF_INET; //选择IPV4
sin.sin_port = htons(4567); //确定服务器端口号
sin.sin_addr.S_un.S_addr = INADDR_ANY;

bind(sockfd, &sin, sizeof(sin))
```

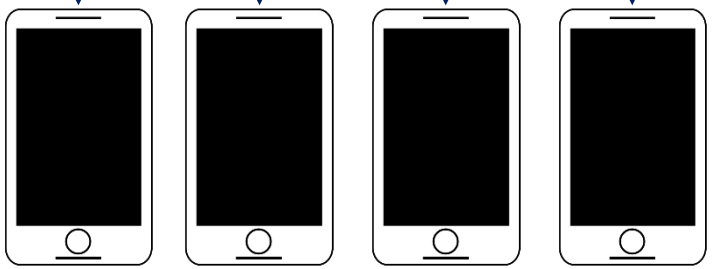


bind()

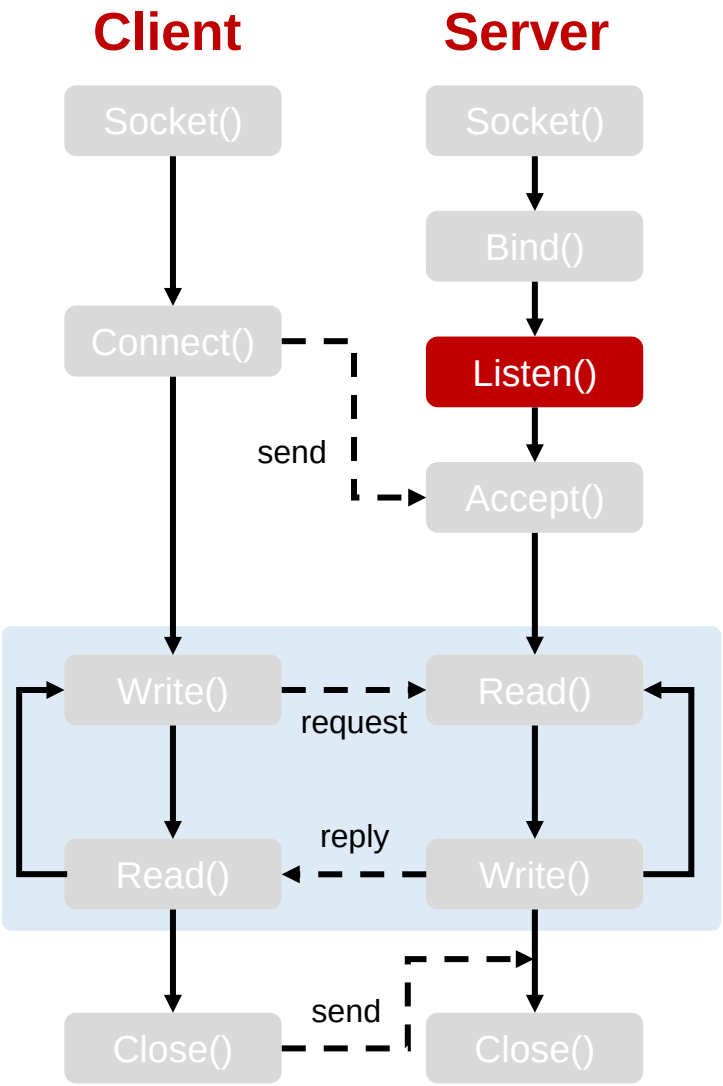
int listen(int sockfd, int backlog);

Socket标识

Sokect最大连接个数



```
listen(sockfd, 4); // allows maximum 4 connections
```

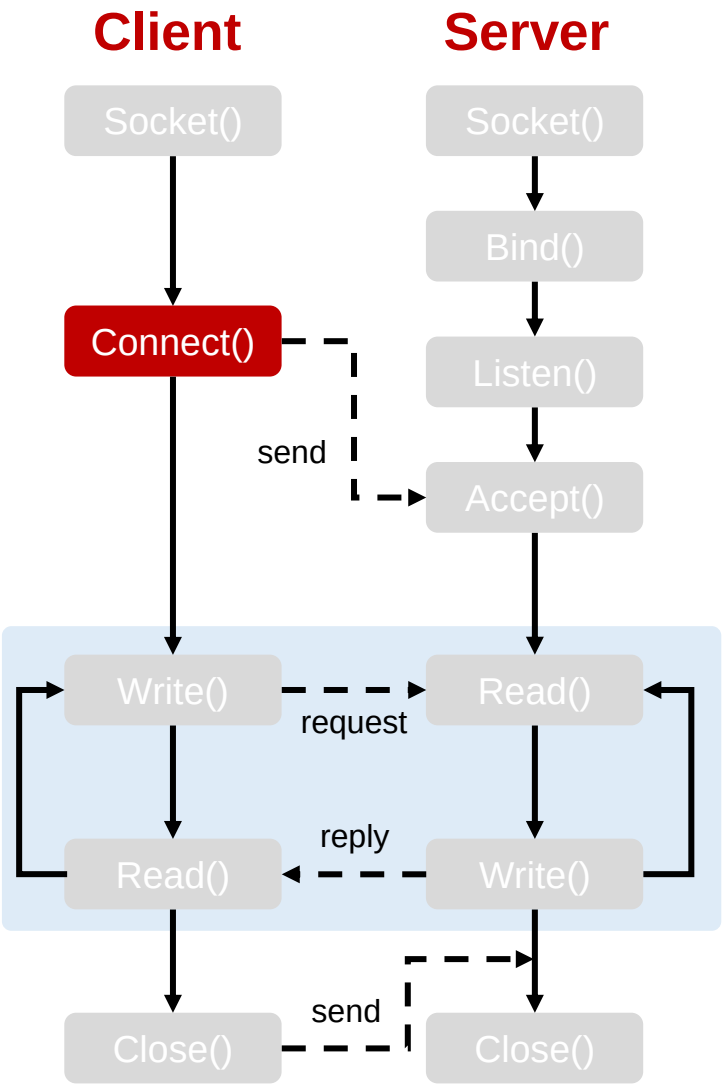


connect()

int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);

- sockfd → Socket标识 (client)
- addrlen → Server地址长度
- addr → Server地址

```
struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET; //选择IPV4
servAddr.sin_port = htons(4567); //确定服务器端口号
servAddr.sin_addr.S_un.S_addr = inet_addr(server_IP); //确定服务器IP地址

connect(sockfd, (const struct sockaddr*)&servAddr, sizeof(servAddr))
```

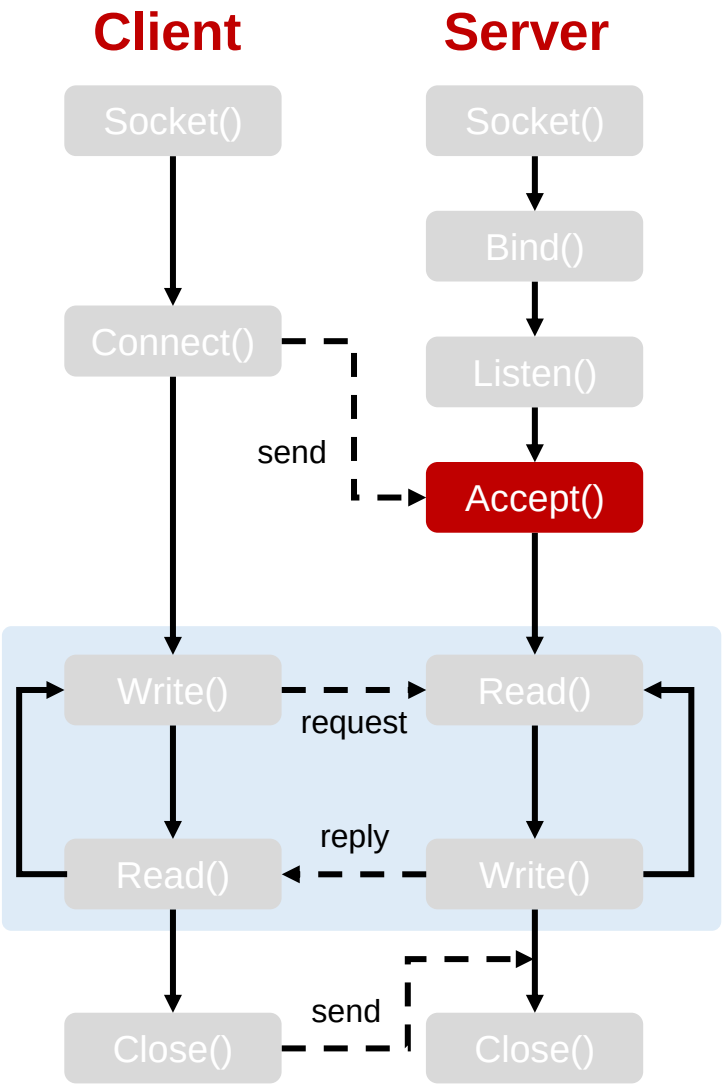


accept()

int accept(int sockfd, struct sockaddr *addr, socklen_t addrlen);

- sockfd → **Socket标识 (server)**
- addrlen → **Client地址长度**
- addr → **Client地址**
- return → **Create a new connected socket**

```
struct sockaddr_in remoteAddr;
int nAddrLen = sizeof(remoteAddr);

SOCKET sClient = 0;
sClient = accept(sockfd, &remoteAddr, &nAddrLen);
```



write() & read()

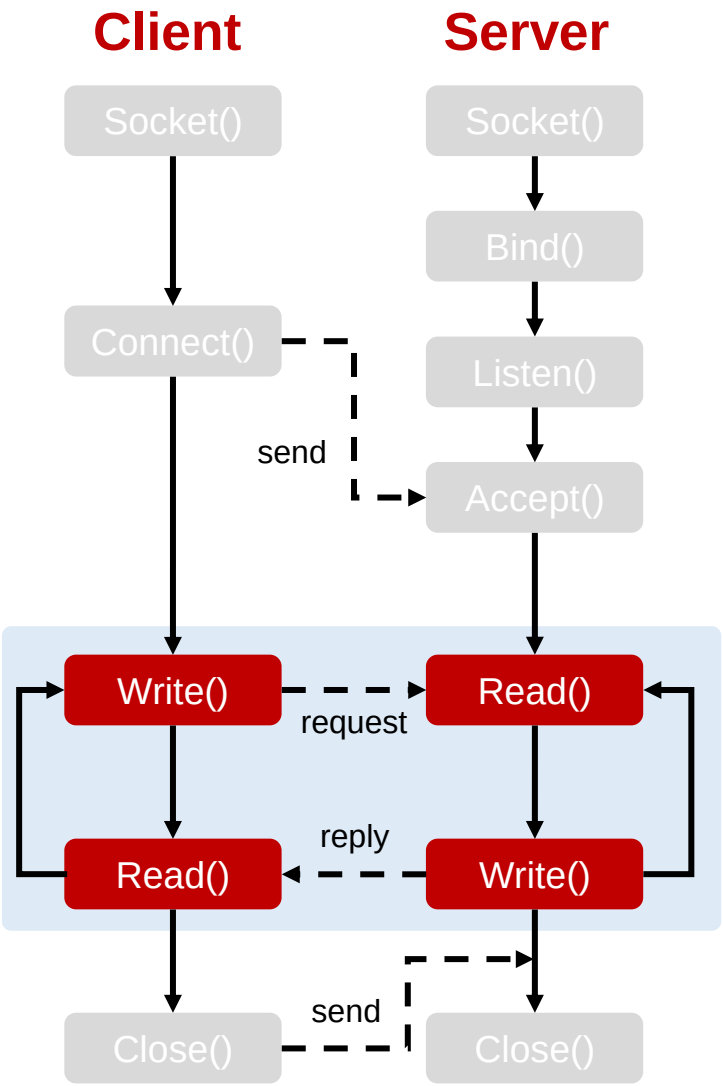
Different functions perform I/O

- recv()/send()

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- recvfrom()/sendto()

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);
```



write() & read()

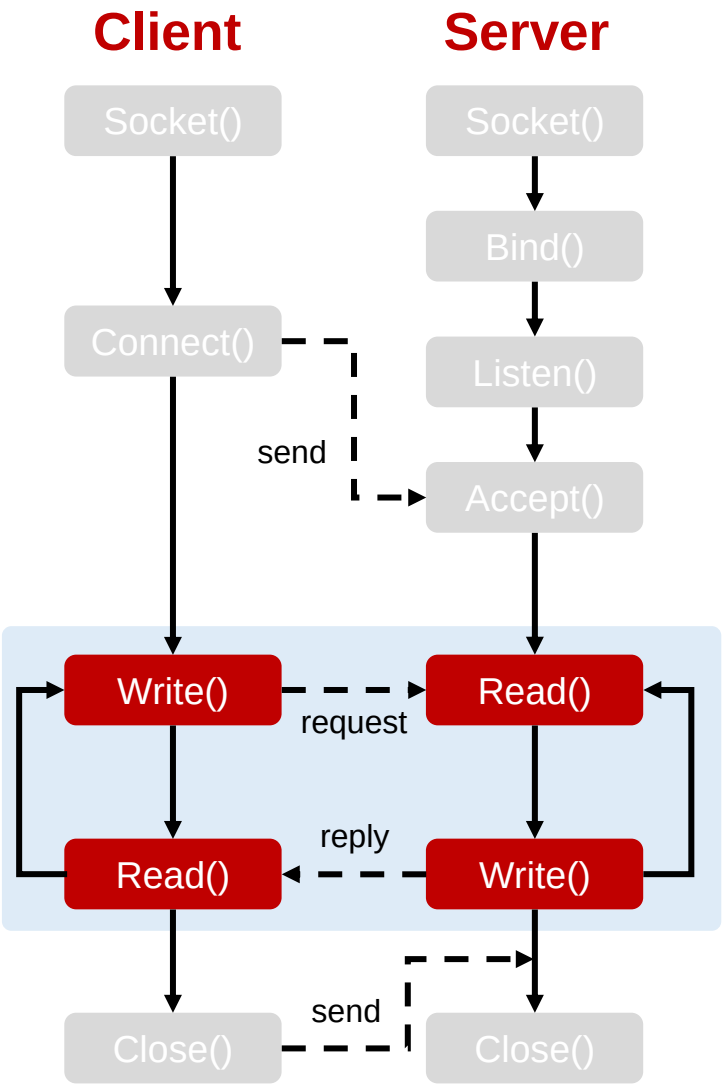
Different functions perform I/O

```
char szText[] = " Hello World! \r\n";
send(sClient, szText, sizeof(szText), 0);

sendto(sockfd, szText, sizeof(szText), 0, (const struct
sockaddr*)&servAddr, sizeof(servAddr));
```

flag = 0, 阻塞
一直发送

```
char buff[256];
int nRecv = recv(sClient, buff, 256, 0);

int length = sizeof(servAddr);
int nRecv = recvfrom(sockfd, buff, 256, 0, (struct
sockaddr*)&servAddr, &length);
```

flag = 0, 阻塞
一直接收



close()

int closesocket(int sockfd);

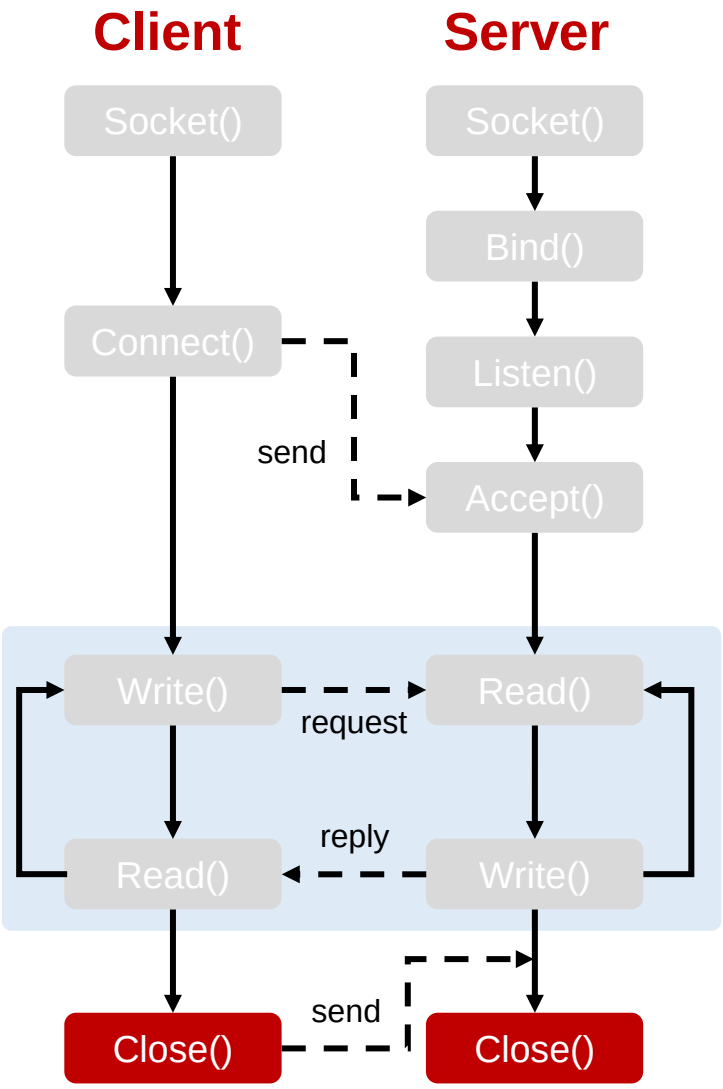
Socket描述字

Server

```
closesocket(sClient);

closesocket(sockfd);
```

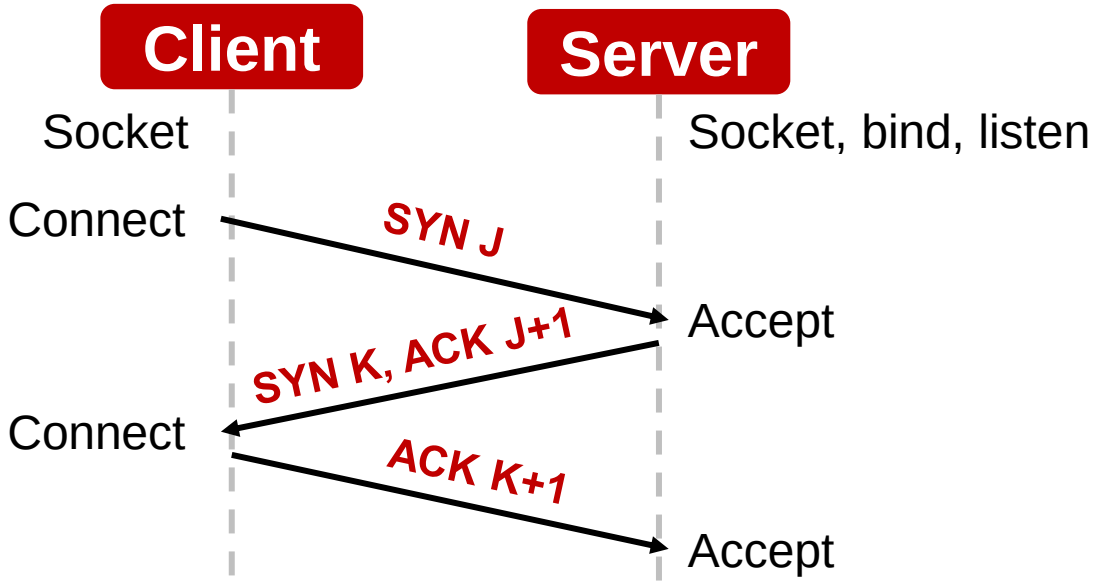
Client

```
closesocket(sockfd);
```



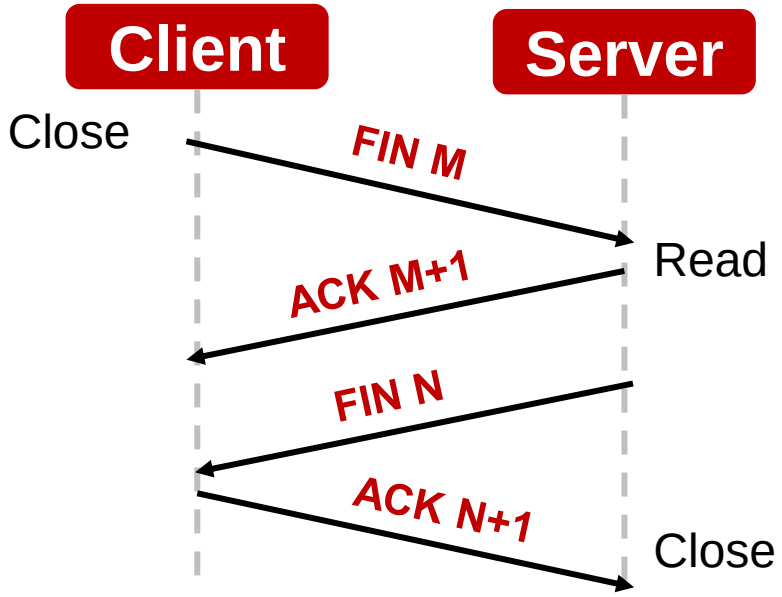
TCP builds/closes connection

TCP builds the connection
3次握手



- Client to server: SYN J
- Server to client: SYN K, ACK J+1
- Client to server: ACK K+1

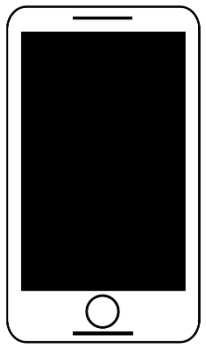
TCP closes the connection
4次握手



- Client to server: FIN M
- Server to client: ACK M+1
- Server to client: FIN N
- Client to server: ACK N+1

Case study: socket

Client.c



Connecting 2 nodes in a network to communicate with each other

Server.c

Case study: Server.c

```c
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <winsock2.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#pragma comment(lib, "WS2_32")

void CInitSock_func(BYTE minorVer, BYTE majorVer)
{
WSADATA wsaData;
WORD sockVersion = MAKEWORD(minorVer, majorVer);
if (WSAStartup(sockVersion, &wsaData) != 0)
{
exit(0);
}
}
int main()
{
CInitSock_func(2, 2);
```

根据版本号加载相应的库文件

选择socket的版本（副版本号，主版本号）

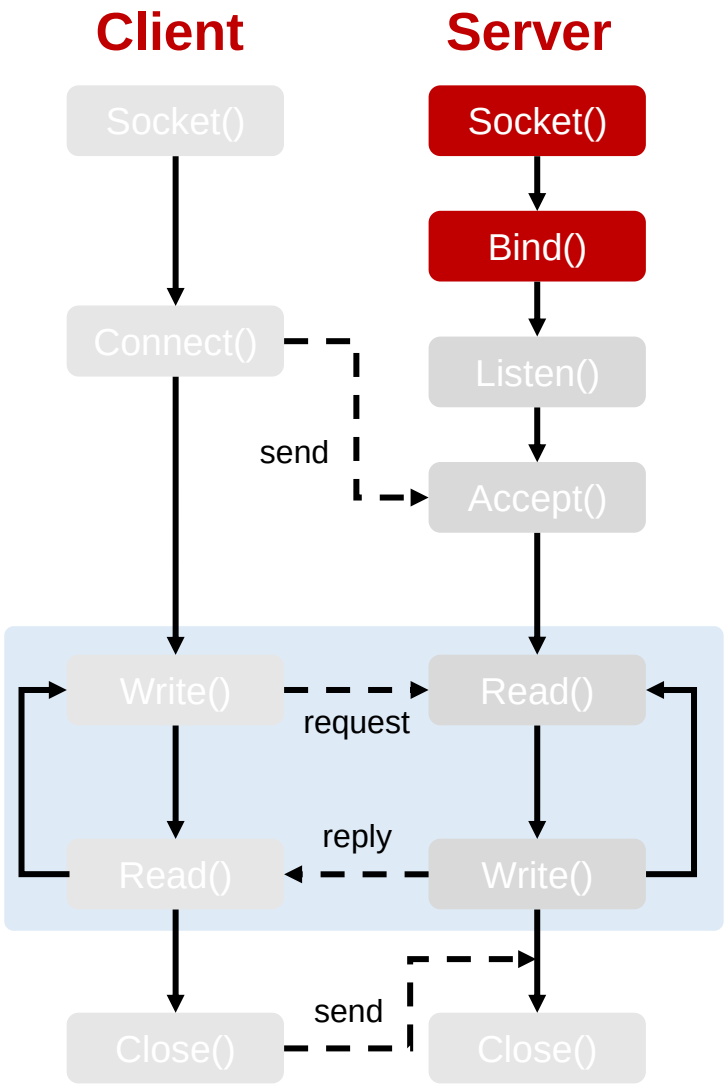
Case study: Server.c

```c
SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sockfd == INVALID_SOCKET)
{
    printf("Failed socket() \n");
    return 0;
}

struct sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(4567);
sin.sin_addr.S_un.S_addr = INADDR_ANY; // IP和端口合并

int flag = bind(sockfd, (const struct sockaddr*)&sin,
sizeof(sin));

if (flag == SOCKET_ERROR)
{
    printf("Failed bind() \n");
    return 0;
}
```

创建一个套接字

定义IP address + port
用户可用端口：1024 ~ 49151

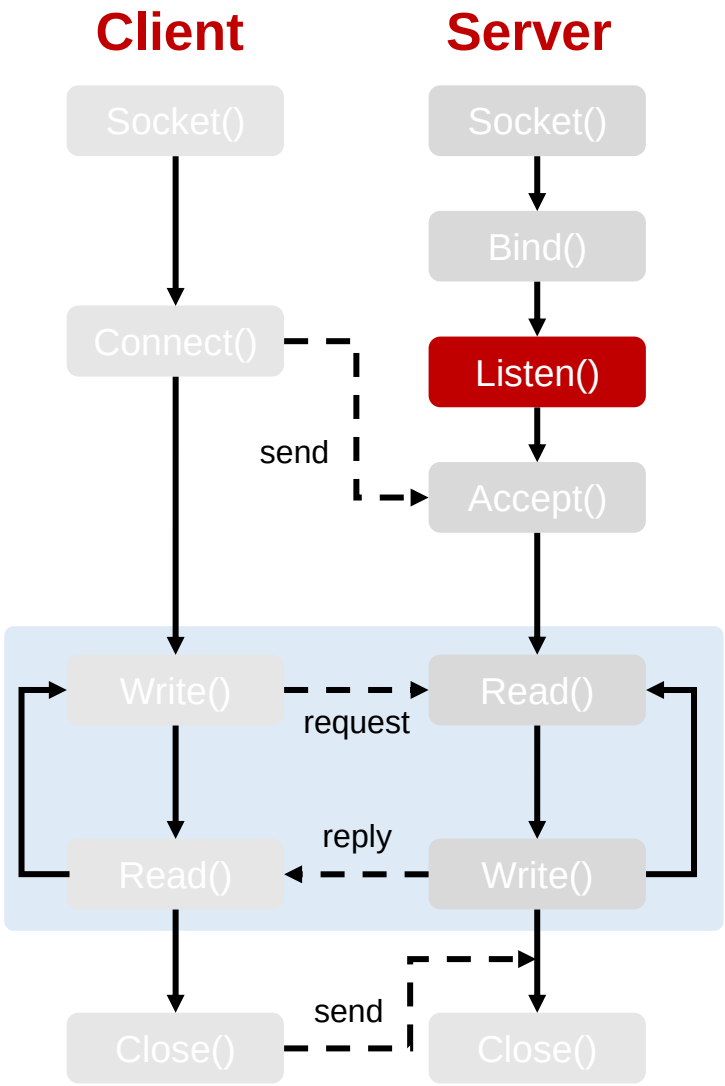
绑定socket和地址 (IP + port)



Case study: Server.c

进入监听模式
2 = 监听队列中允许保持的尚未处理的最大连接数

```
flag = listen(sockfd, 2);

if (flag == SOCKET_ERROR)
{
    printf("Failed listen() \n");
    return 0;
}
```

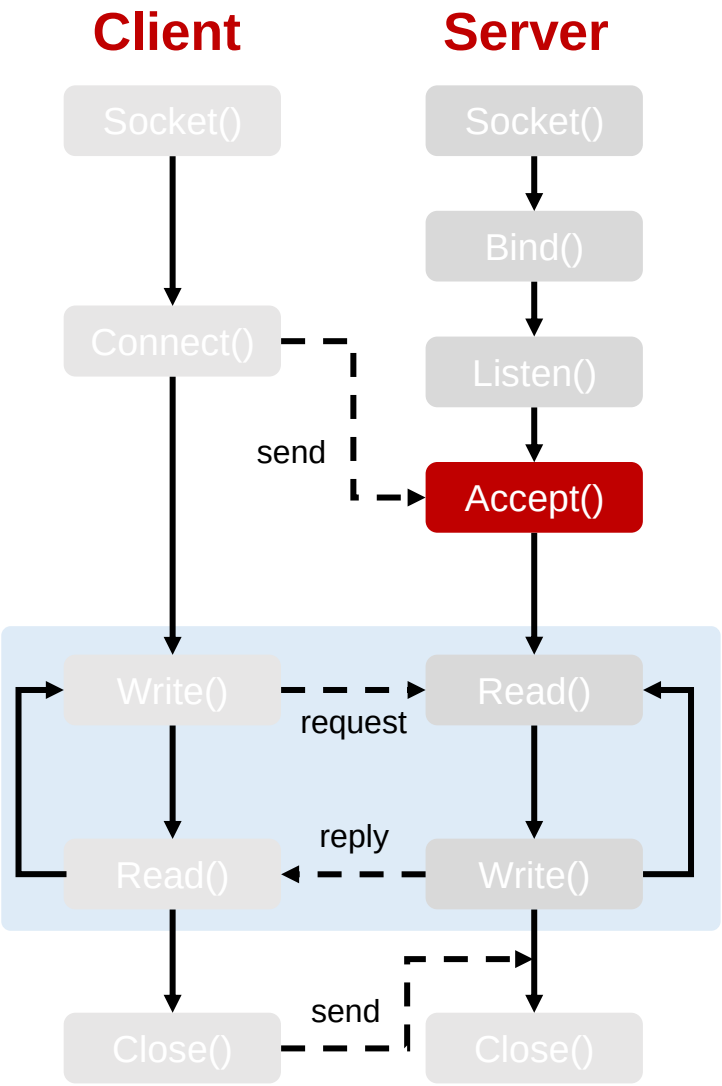


Case study: Server.c

接受用户的连接请求（阻塞）
阻塞：如果没有收到请求，程序会一直停留在这里等待

```
struct sockaddr_in remoteAddr;
int nAddrLen = sizeof(remoteAddr);


SOCKET sClient = accept(sockfd, &remoteAddr, &nAddrLen);
if (sClient == INVALID_SOCKET)
{
    printf("Failed accept()");
    return 0;
}
printf("接受到一个连接：%s \r\n", inet_ntoa(remoteAddr.sin_addr));
```

获得客户端IP地址



Case study: Server.c

```
while (1)
{
    char buff[256];
    int nRecv = recv(sClient, buff, 256, 0);
    if (nRecv > 0)
    {
        printf(" 接收到数据: %s\n", buff);
        send(sClient, buff, sizeof(buff), 0);
    }
}

closesocket(sClient);

closesocket(sockfd);
```

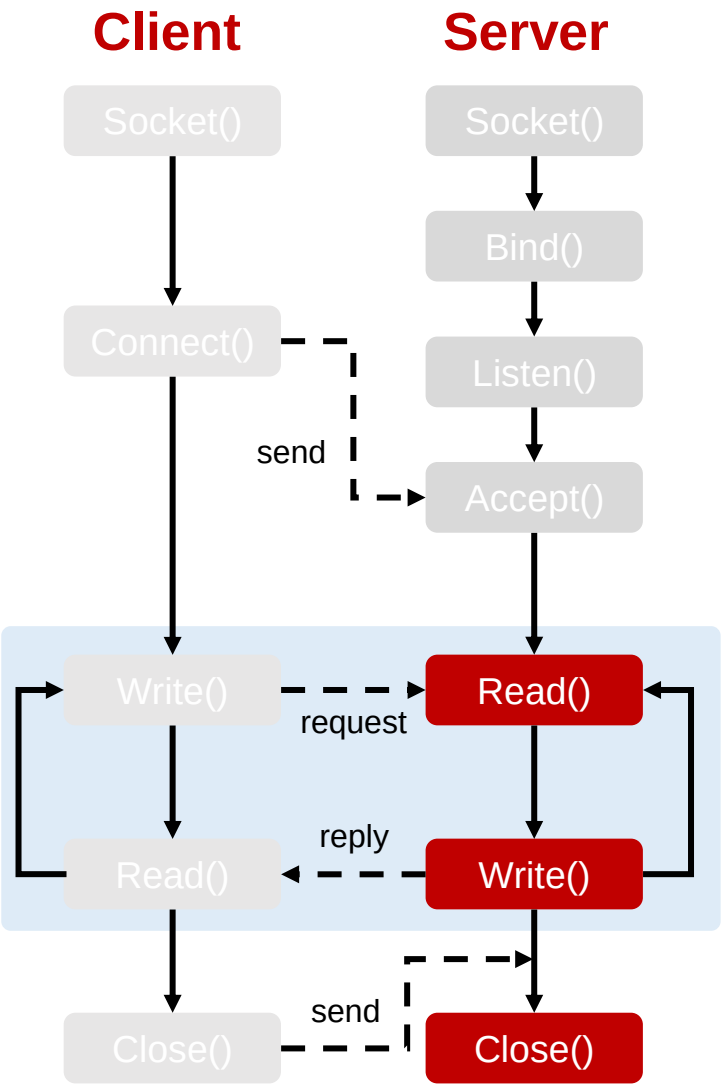
接收客户端的数据
阻塞：一直接收

长度

向客户端发送数据
阻塞：一直发送

关闭client socket

关闭server socket



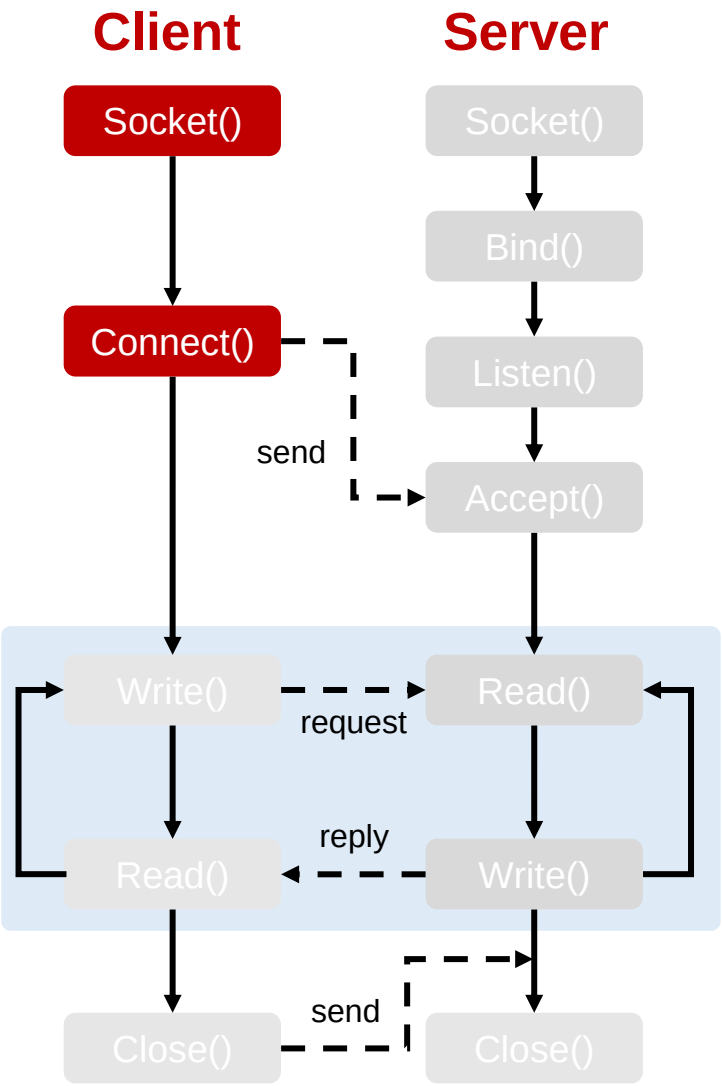
Case study: Client.c

```
SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sockfd == INVALID_SOCKET)
{
    printf(" Failed socket() \n");
    return 0;
}

struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(4567);
servAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");

if (connect(sockfd, (const struct sockaddr*)&servAddr, sizeof(servAddr))
== -1)
{
    printf(" Failed connect() \n");
    return 0;
}
```

创建套接字

设置服务器网址
127. 0. 0. 1 为本地环回
自己发给自己

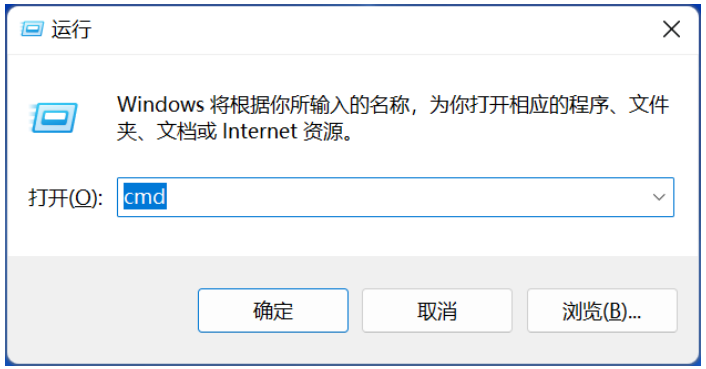
连接服务器



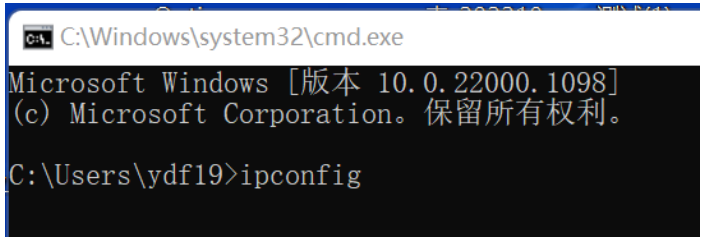
Case study: Client.c

如何连接另一台电脑 ？

①键盘上输入 win + R



②输入 cmd 确定



③输入ipconfig

如果你用的无线网，就记下这个IPV4地址！！！

Case study: Client.c

```
SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sockfd == INVALID_SOCKET)
{
printf(" Failed socket() \n");
return 0;
}

struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(4567);
servAddr.sin_addr.S_un.S_addr = inet_addr("192.168.3.75");

if (connect(sockfd, (const struct sockaddr*)&servAddr, sizeof(servAddr))
== -1)
{
printf(" Failed connect() \n");
return 0;
}
```

把IPV4地址填在这里
就可以连接这台电脑

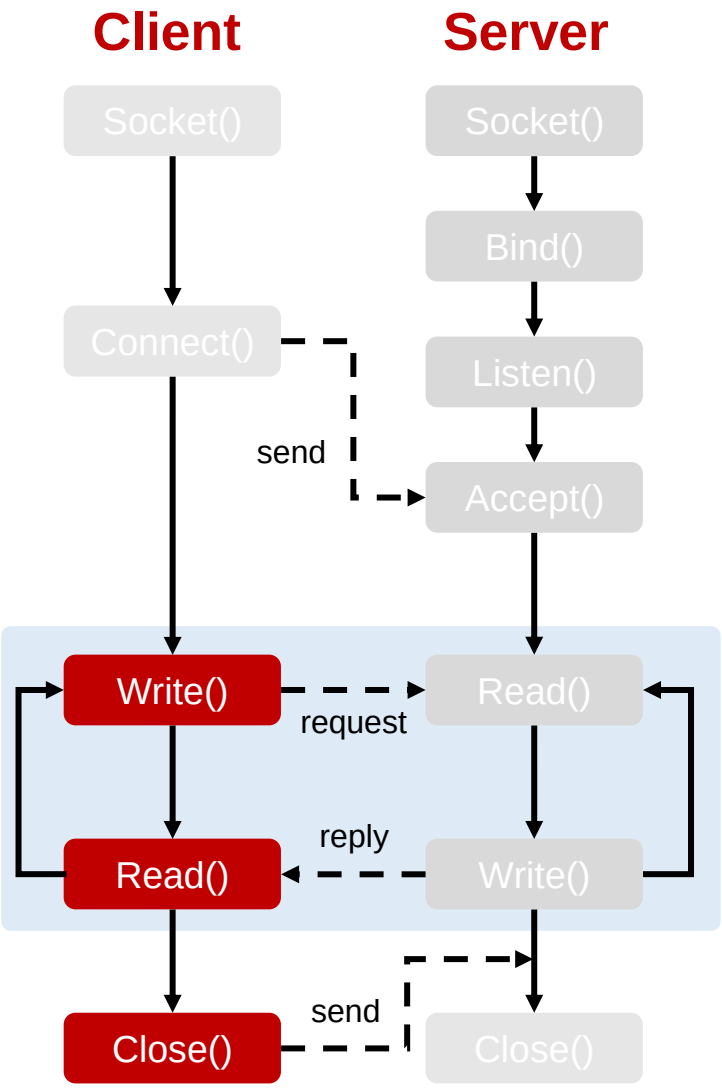
Case study: Client.c

```
char buff[256];
char szText[256];
while (TRUE)
{
     int length = sizeof(servAddr);
     int nRecv = recvfrom(sockfd, buff, 256, 0, (struct sockaddr*)&servAddr,
     &length);
     if (nRecv > 0)
     {
          printf("接收到数据: %s\n", buff);
     }
     gets_s(szText, 256); // user input
     sendto(sockfd, szText, sizeof(szText), 0, (const struct sockaddr*)&servAddr,
     sizeof(servAddr));
}

closesocket(sockfd);
```

从服务器接收数据

向服务器发送数据

关闭套接字



作业遇到的问题！

Design before implement (step-by-step)

1. Read the bin file using fread (client.c)
2. Sort the struct using insertion sort (client.c)
3. Create socket connection between client and server (client.c and server.c)
4. Send sorted struct from client to server by char* (server.c)
5. Receive sorted struct and write to CSV file using fprintf (server.c)

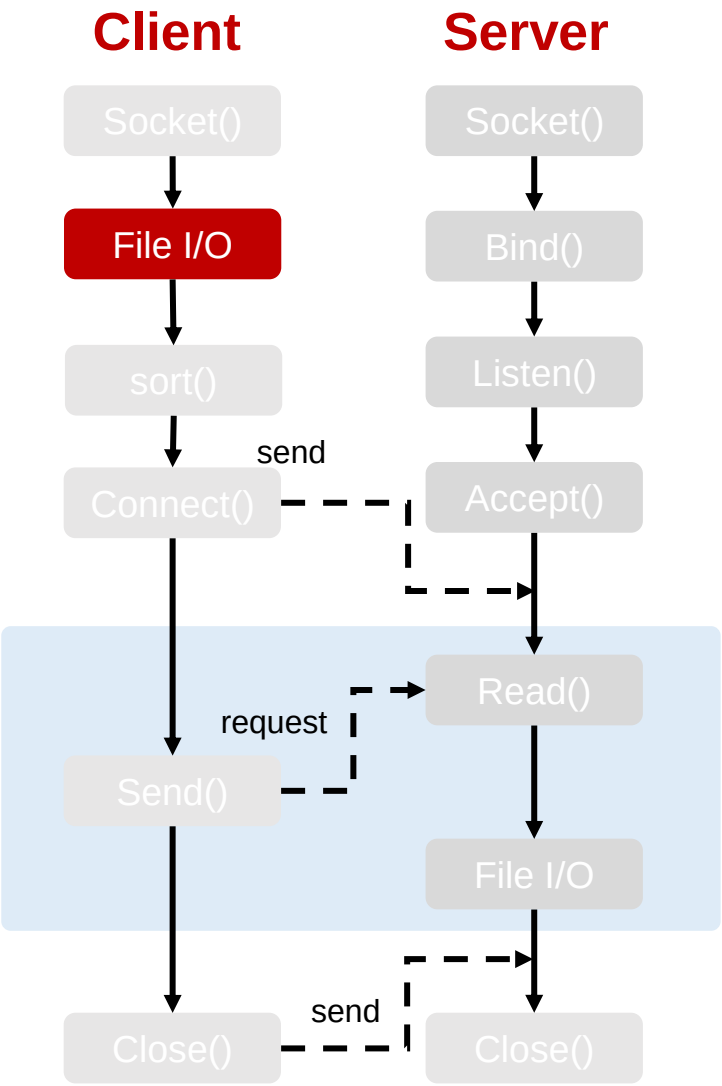
作业：Client.c

```c
typedef struct
{
int stu_number;
char name[20];
int score;
}score;

score Transcripts[10];
FILE* fptr;
fopen_s(&fptr,"C://Users//ydf19//Desktop/
/data.bin", "rb");
fread(Transcripts, sizeof(score), 10,
fptr);
insertion_sort(Transcripts, 10);
for (int i = 0; i < 10; i++)
{
    printf("%d,%s,%d\n",
    Transcripts[i].stu_number,
    Transcripts[i].name,
    Transcripts[i].score);
}
```

使用VS的同学，fopen有可能报错，根据提示可以更换为fopen_s

fopen_s把fopen的返回值变成了参数，根据提示这里需要一共FILE**类型的参数

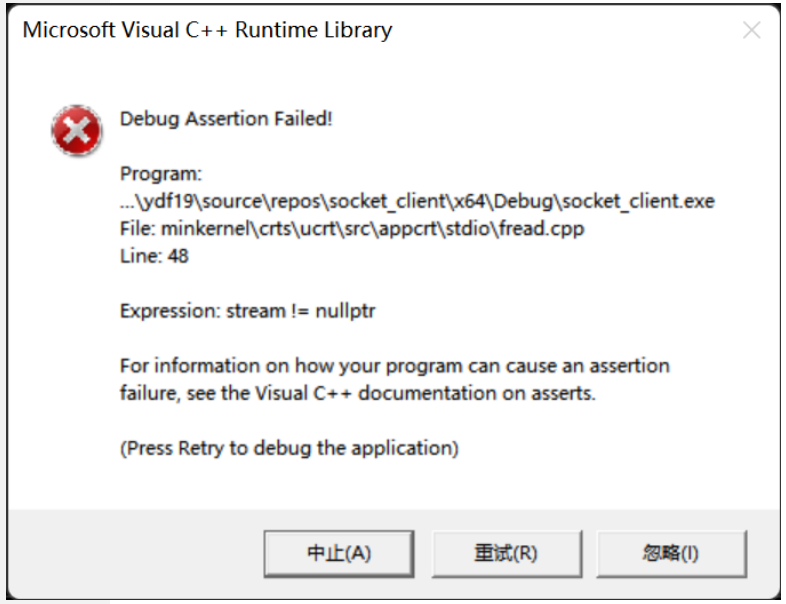
这里写绝对路径，确保程序可以找到bin文件双斜杠//

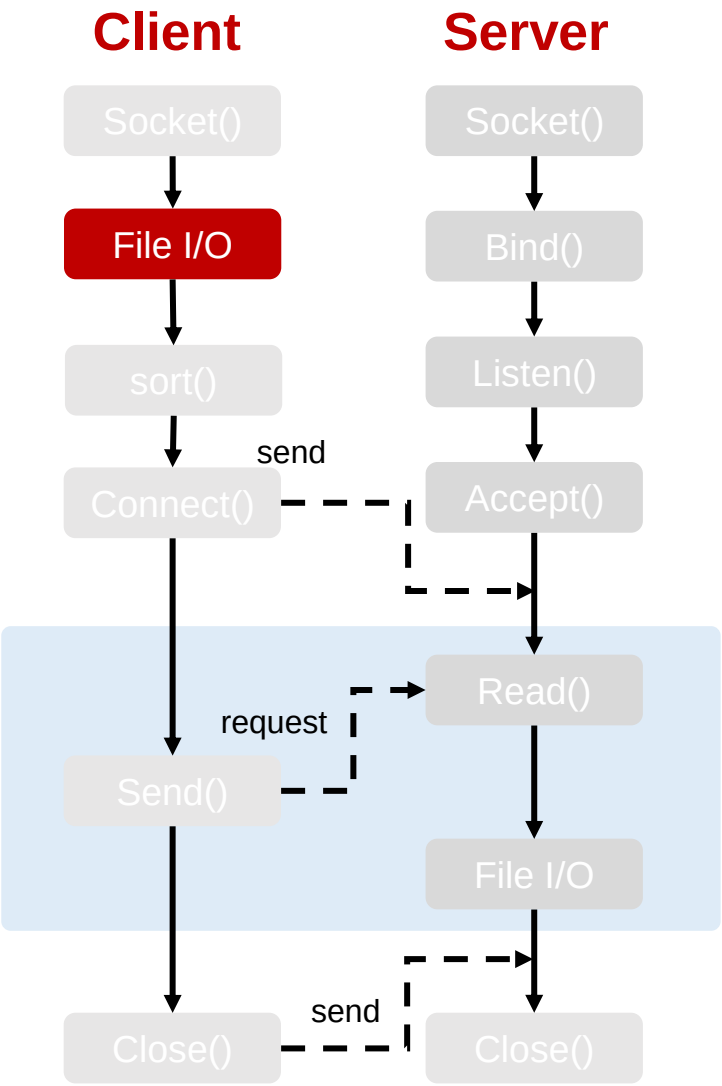


作业：Client.c

```c
typedef struct
{
int stu_number;
char name[20];
int score;
}score;
score Transcripts[10];
FILE* fptr;
fopen_s(&fptr, "data.bin", "rb");
fread(Transcripts, sizeof(score),
fptr);
insertion_sort(Transcripts, 10);
for (int i = 0; i < 10; i++)
{
printf("%d,%s,%d\n",
Transcripts[i].stu_number,
Transcripts[i].name,
Transcripts[i].score);
}
```

这里如果想这么写，必须把bin文件和代码放在一起，否则会报错

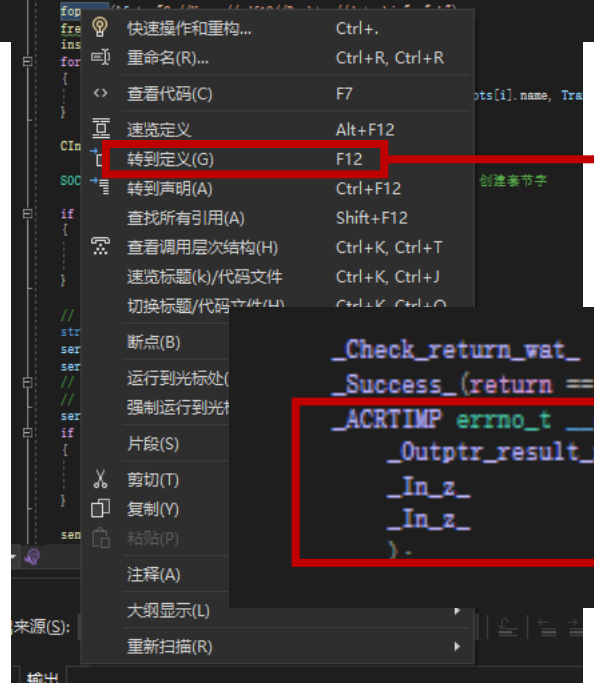




作业：Client.c

```c
int main()
{
    score Transcripts[10];
    FILE* fptr;
    fopen_s(&fptr, "C://Users//ydf19//Desktop//data.bin", "rb");
    fread(Transcripts, sizeof(score), 10, fptr);
    insertion_sort(Transcripts, 10);
    for (int i = 0; i < 10; i++)
    {
        printf("%d,%s,%d\n", Transcripts[i].stu_number, Transcripts[i].name, Transcripts[i].score);
```

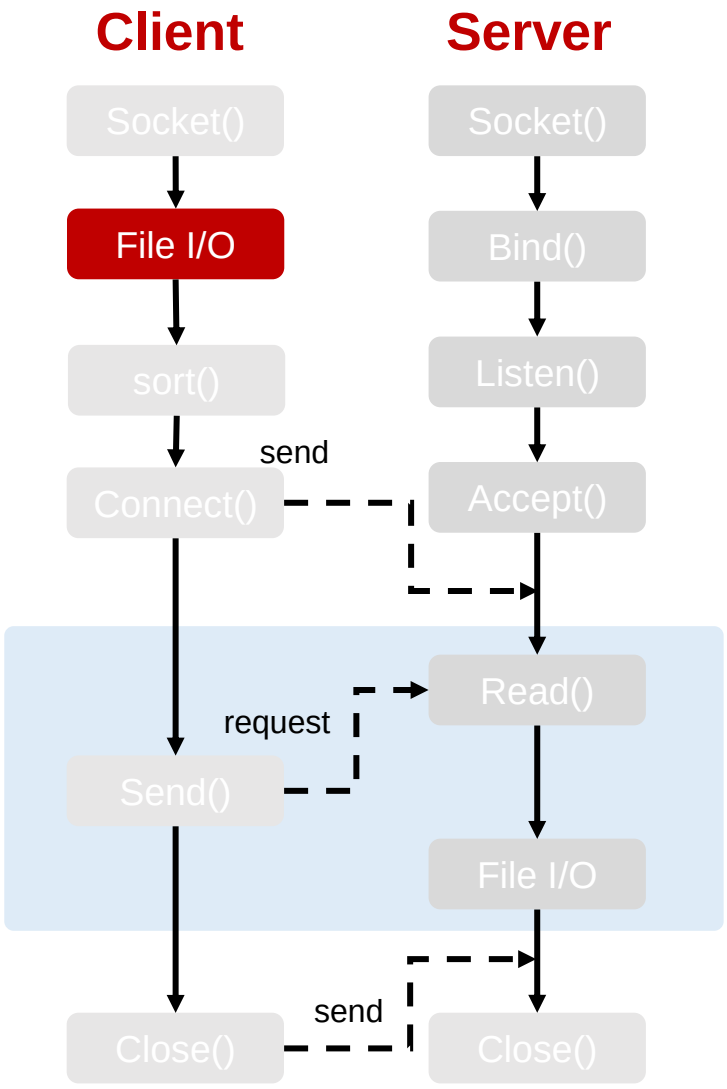
双击选中这个函数，点右键



点击转到定义

```c
_Check_return_wat_
_Success_(return == 0)
_ACRTIMP errno_t __cdecl fopen_s(
    _Outptr_result_nullonfailure_ FILE**       _Stream,
    _In_z_                        char const* _FileName,
    _In_z_                        char const* _Mode
    ).
```

在这里你可以看到函数
需要什么类型的参数



作业：Client.c

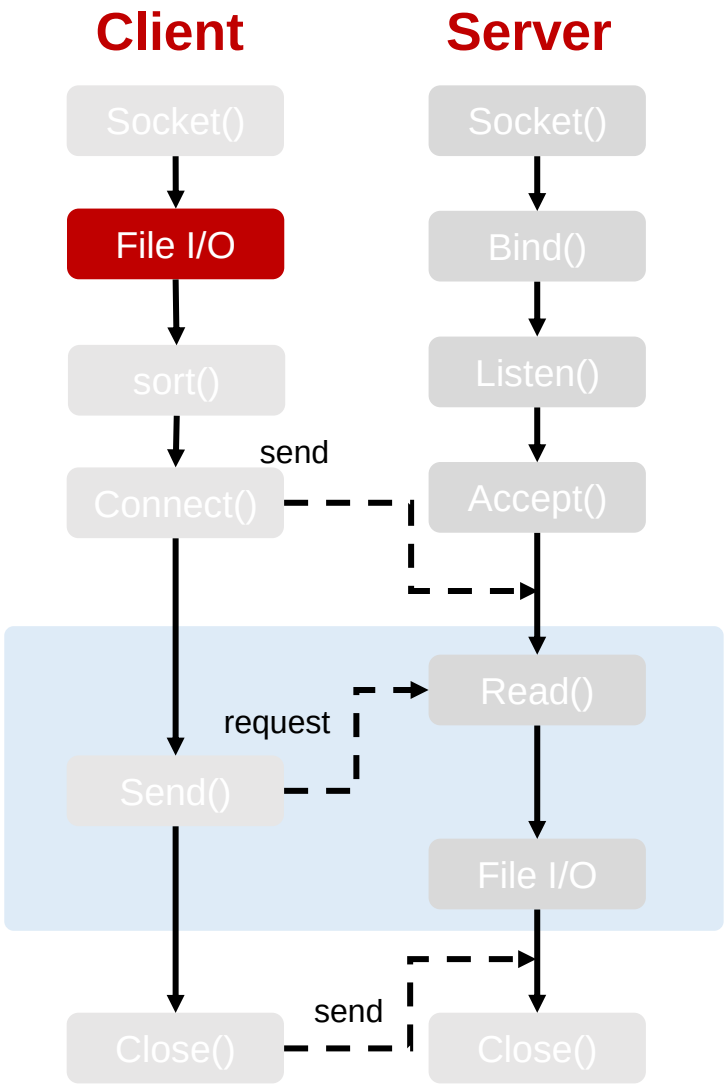
```c
typedef struct
{
int stu_number;
char name[20];
int score;
}score;

score Transcripts[10];
FILE* fptr;
fopen_s(&fptr,"C://Users//ydf19//Desktop/
/data.bin", "rb");
fread(Transcripts, sizeof(score), 10,
fptr);
insertion_sort(Transcripts, 10);
for (int i = 0; i < 10; i++)
{
    printf("%d,%s,%d\n",
    Transcripts[i].stu_number,
    Transcripts[i].name,
    Transcripts[i].score);
}
```

如果写成“wb”, bin文件会被清空；即使再改回“rb”，读取到的也是乱码， 需要重新下载bin

每个结构体的大小，读取10次

```c
fread(Transcripts, sizeof(score) * 10, 1,
fptr);
```

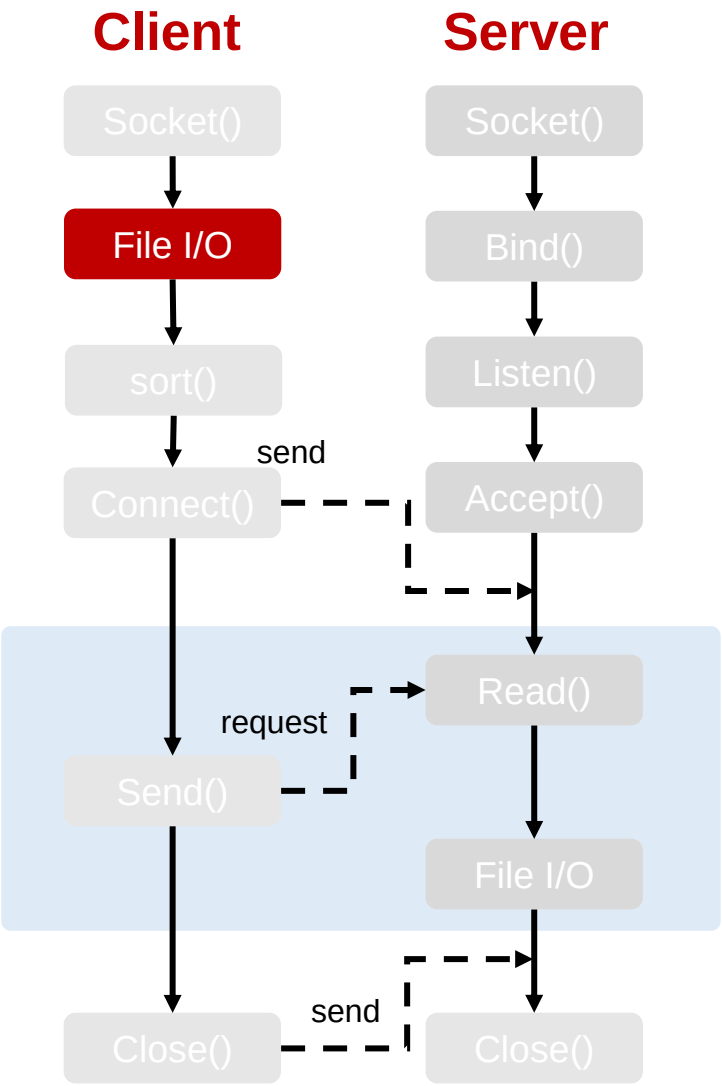
```c
fread(Transcripts, sizeof(Transcripts),
1, fptr);
```



作业：Client.c

```c
typedef struct
{
int stu_number;
char name[20];
int score;
}score;
score Transcripts[10];
FILE* fptr;
fopen_s(&fptr,"C://Users//ydf19//Desktop/
/data.bin", "rb");
fread(Transcripts, sizeof(score), 10,
fptr);
insertion_sort(Transcripts, 10);
for (int i = 0; i < 10; i++)
{
    printf("%d,%s,%d\n",
    Transcripts[i].stu_number,
    Transcripts[i].name,
    Transcripts[i].score);
}
```

每进行一步操作可以打印一遍，如果有BUG方便定位哪里出错



作业：Client.c

```
46
47  int main()
48  {
49      score Transcripts[10];
50      FILE* fptr;
51      fopen_s(&fptr, "C://Users//ydf19//Desktop//data.bin", "rb");
52      fread(Transcripts, sizeof(score), 10, fptr);
53      insertion_sort(Transcripts, 10);
54      for (int i = 0; i < 10; i++)
55      {
56          printf("%d,%s,%d\n", Transcripts[i].stu_number, Transcripts[i].name, Transcripts[i].score);
57      }
58
59      CInitSock(2, 2);//设置版本信息
60
61      SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);// 创建套节字
62
63      if (sockfd == INVALID_SOCKET)
64      {
65          printf(" Failed socket() \n");
66          return 0;
67      }
```

79 %　0　1

监视 1

搜索(Ctrl+E)　搜索深度: 3

名称	值	类型
Transcripts	0x000000fda6eff470 {{stu_number=7 name=0x000000fda6eff474 "David" score=99 },...	score[10]
[0]	{stu_number=7 name=0x000000fda6eff474 "David" score=99 }	score
stu_number	7	int
name	0x000000fda6eff474 "David" 查看	char[20]
score	99	int
[1]	{stu_number=3 name=0x000000fda6eff490 "Steve" score=98 }	score
stu_number	3	int
name	0x000000fda6eff490 "Steve" 查看	char[20]
score	98	int
[2]	{stu_number=5 name=0x000000fda6eff4ac "Emma" score=97 }	score
[3]	{stu_number=2 name=0x000000fda6eff4c8 "Tim" score=95 }	score
[4]	{stu_number=4 name=0x000000fda6eff4e4 "Mara" score=94 }	score
[5]	{stu_number=1 name=0x000000fda6eff500 "Jack" score=93 }	score
[6]	{stu_number=8 name=0x000000fda6eff51c "Tyler" score=92 }	score

在这里打一个断点，找BUG会更加方便

这里会显示每一个变量的值

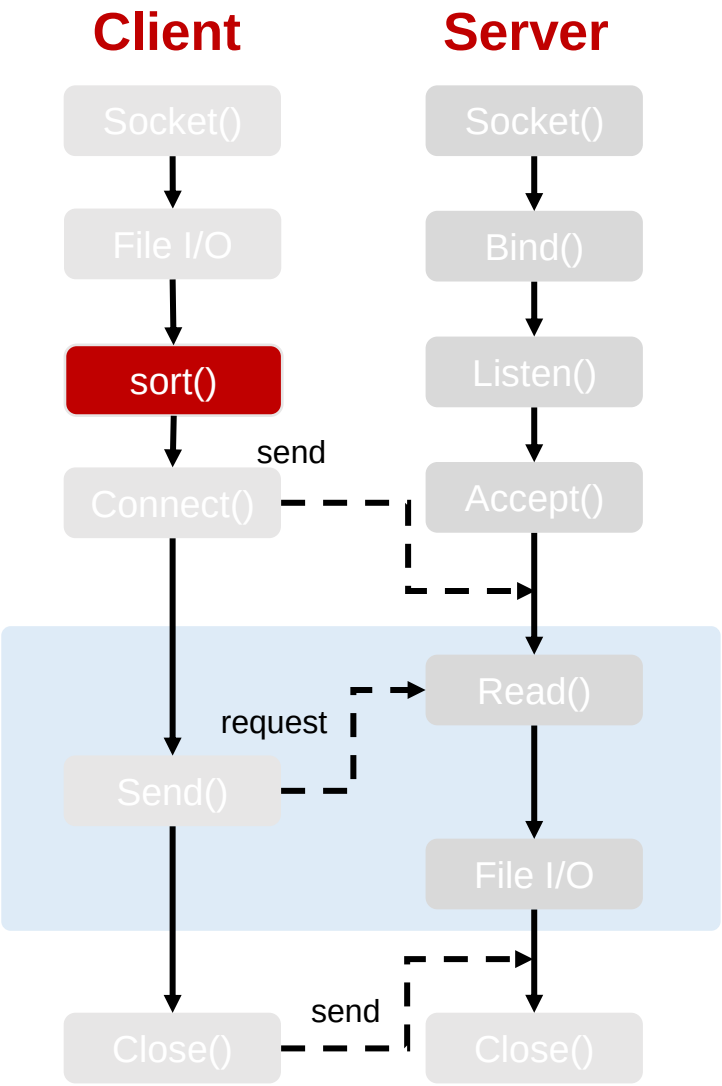
作业：Client.c

```c
void insertion_sort(score* ptr, int length)
{
for (int i = 1; i < length; i++)
{
    for (int j = i; j > 0; j--)
    {
        if (ptr[j].score > ptr[j - 1].score)
        {
        score tempp = ptr[j];
        ptr[j] = ptr[j - 1];
        ptr[j - 1] = tempp;
        }
    }
}
}
```

传入结构体数组的指针（数组中第一个结构体的地址）

知道第一个结构体的位置和一共有几个结构体就可以访问整个结构体数组

排序时交换结构体位置，这样就可以对结构体进行排序



作业：Client.c

```
struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(4567);

servAddr.sin_addr.S_un.S_addr =
inet_addr("127.0.0.1");

if (connect(sockfd, (const struct
sockaddr*)&servAddr, sizeof(servAddr)) ==
-1)
{
printf(" Failed connect() \n");
return 0;
}
```

这些是固定写法，用的时候直接照抄就可以

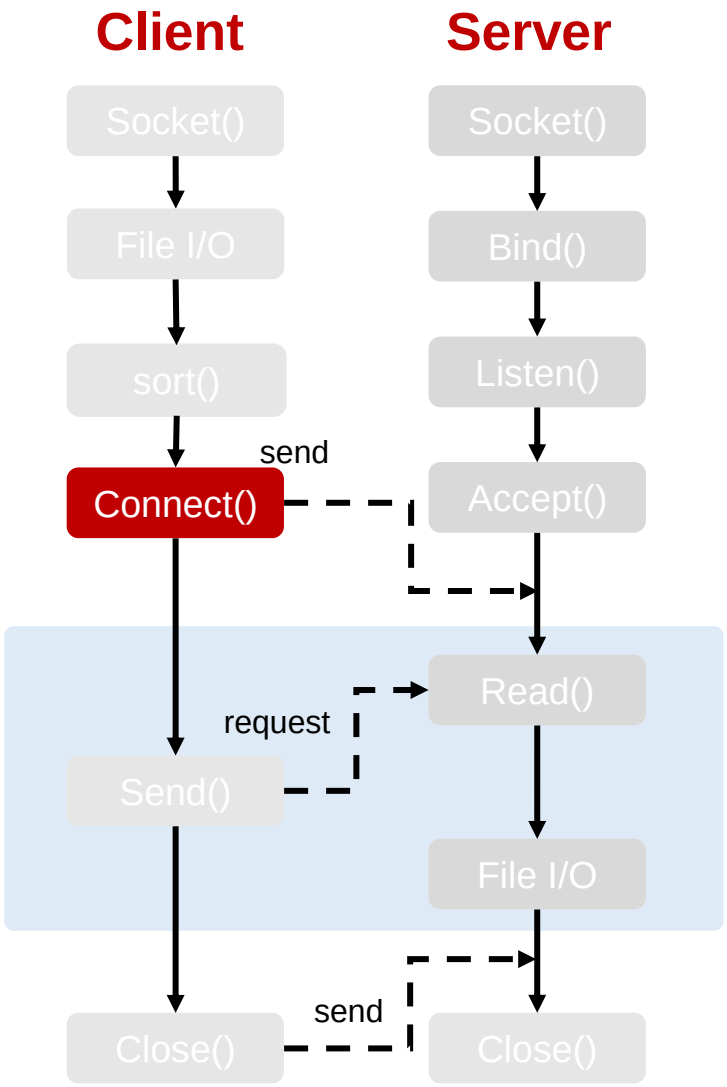
这里有的电脑可能需要强制类型转换，编译器会提示你需要转换成什么类型，以下几种都有可能

(const struct sockaddr*)
(const sockaddr*)
(sockaddr*)

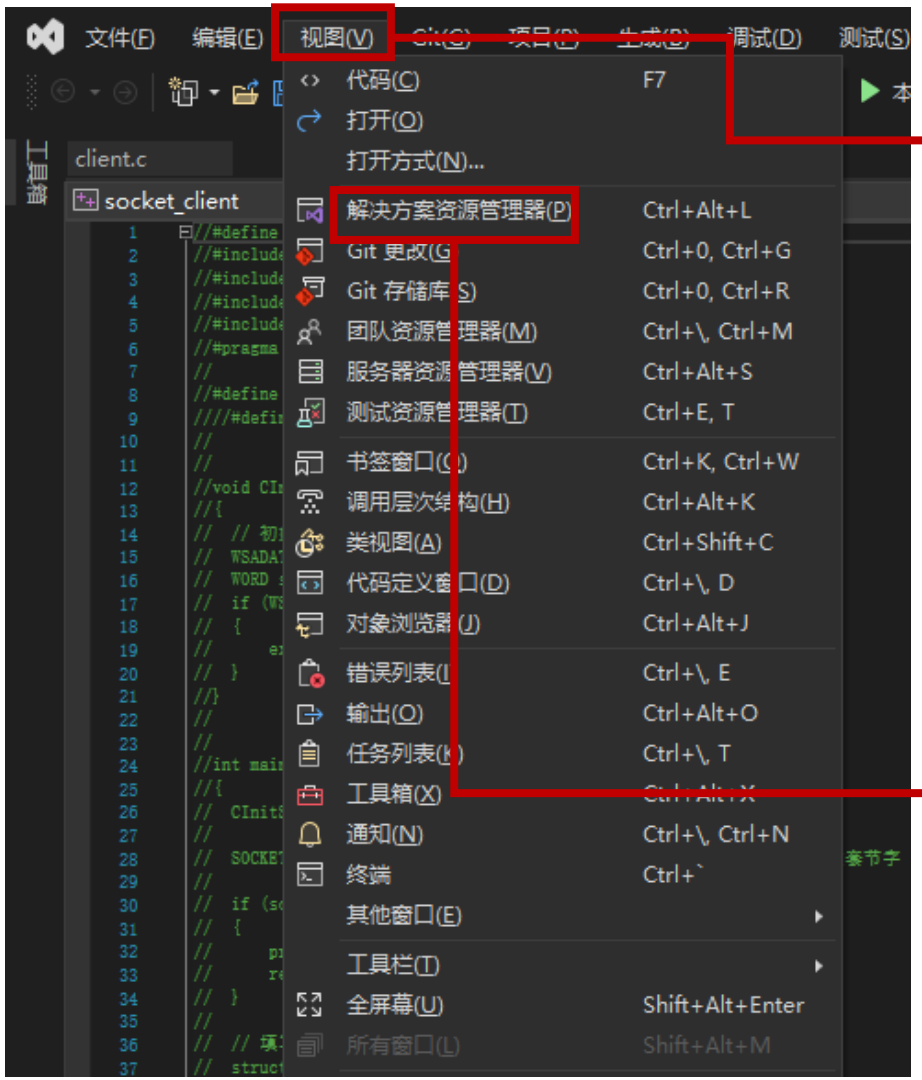
"SOCKET accept(SOCKET,sockaddr *,int *)": 无法将参数 2 从 'sockaddr_in *' 转换为 'sockaddr *'

这是你输入的参数类型

这是函数想要的参数类型

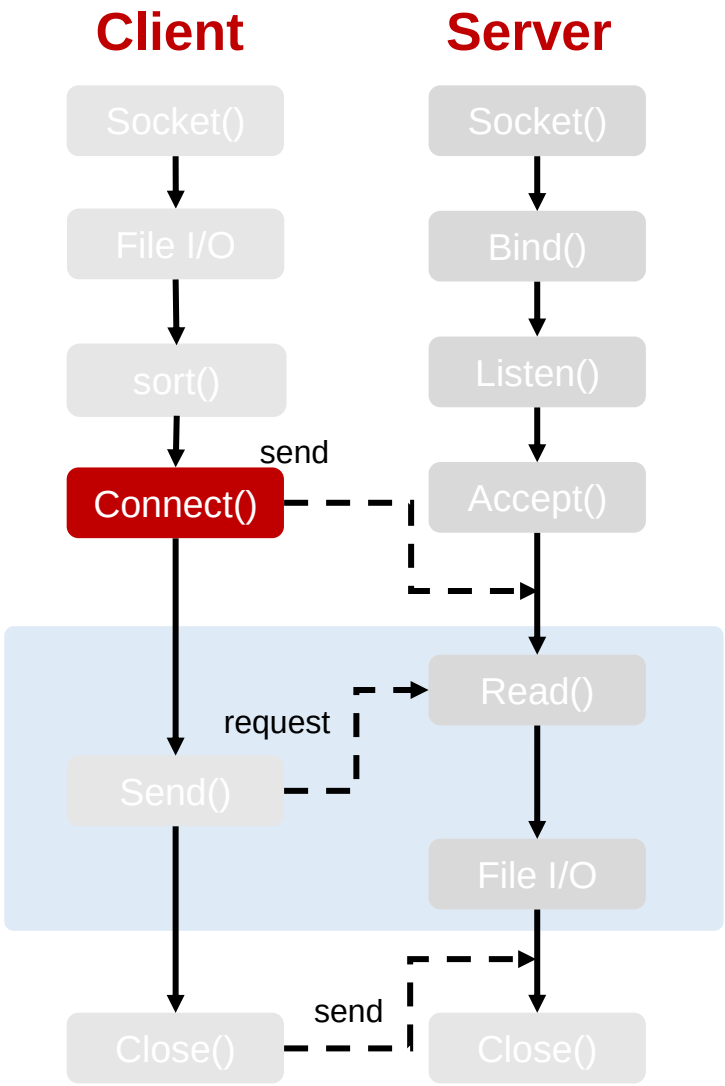


作业：Client.c

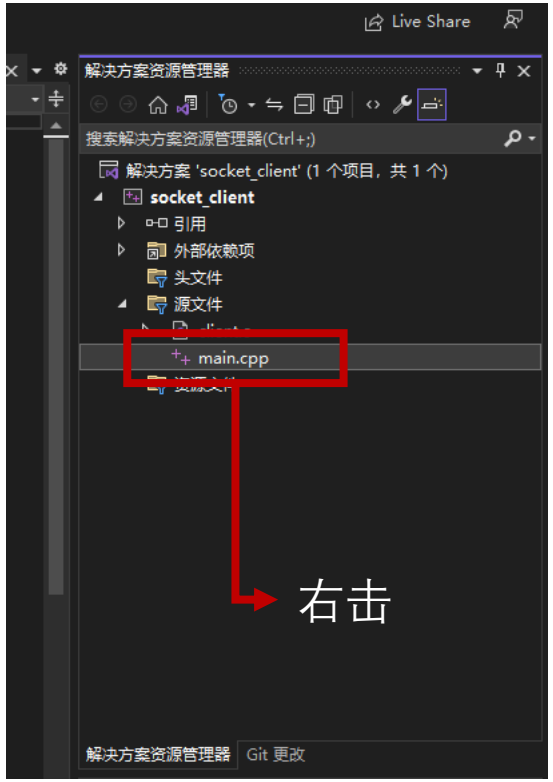


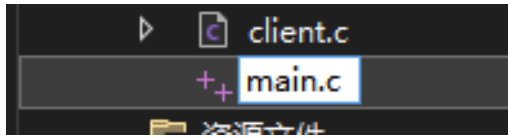
点击视图

点击解决方案管理器

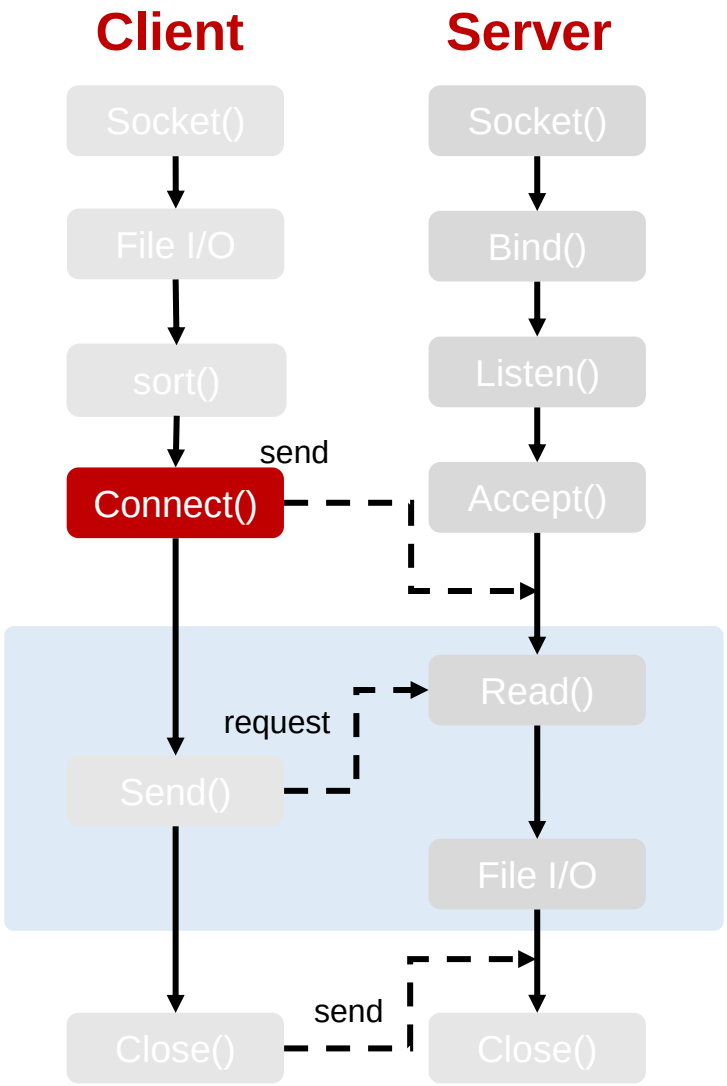


作业：Client.c





把cpp改成c，编译器就不会和你计较刚才的问题



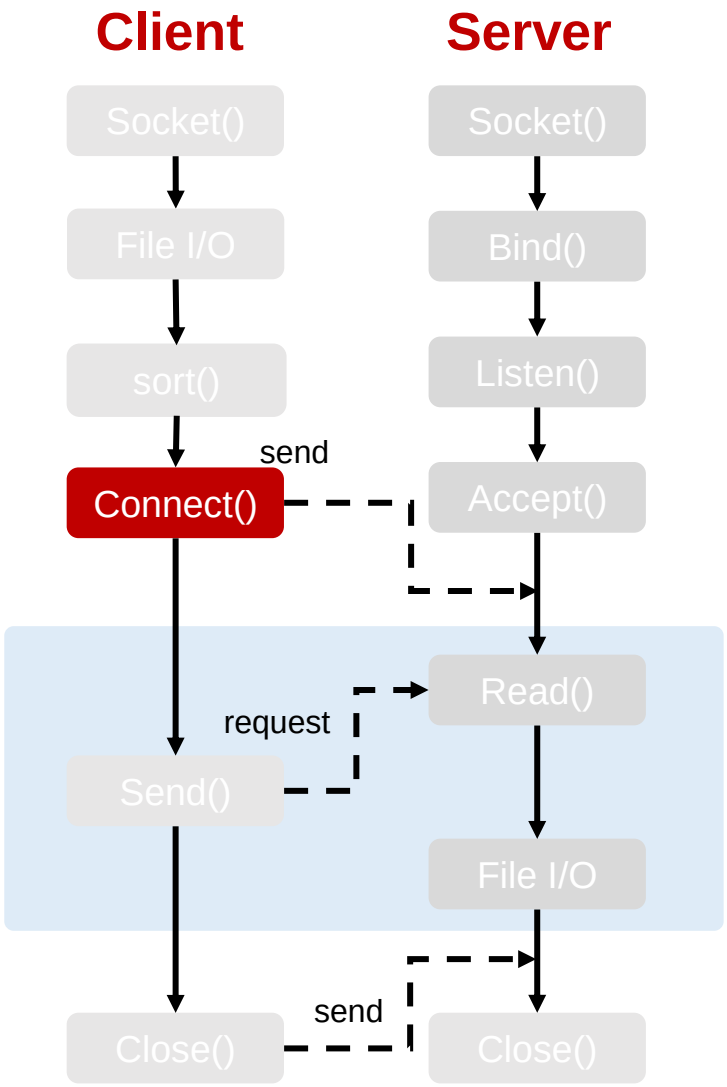
作业：Client.c

```
struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(4567);

servAddr.sin_addr.S_un.S_addr =
inet_addr("127.0.0.1");

if (connect(sockfd, (const struct
sockaddr*)&servAddr, sizeof(servAddr)) ==
-1)
{
printf(" Failed connect() \n");
return 0;
}
```

```
servAddr.sin_addr.S_un.S_addr
= inet_addr("192.168.3.75");
```

这里填写本机的真实IP地址也是可以的

```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . :
   本地链接 IPv6 地址. . . . . : fe80::701a:d780:be90:c147%19
   IPv4 地址 . . . . . : 192.168.3.75
   子网掩码 . . . . . : 255.255.255.0
   默认网关. . . . . : 192.168.3.1
```



作业：Client.c

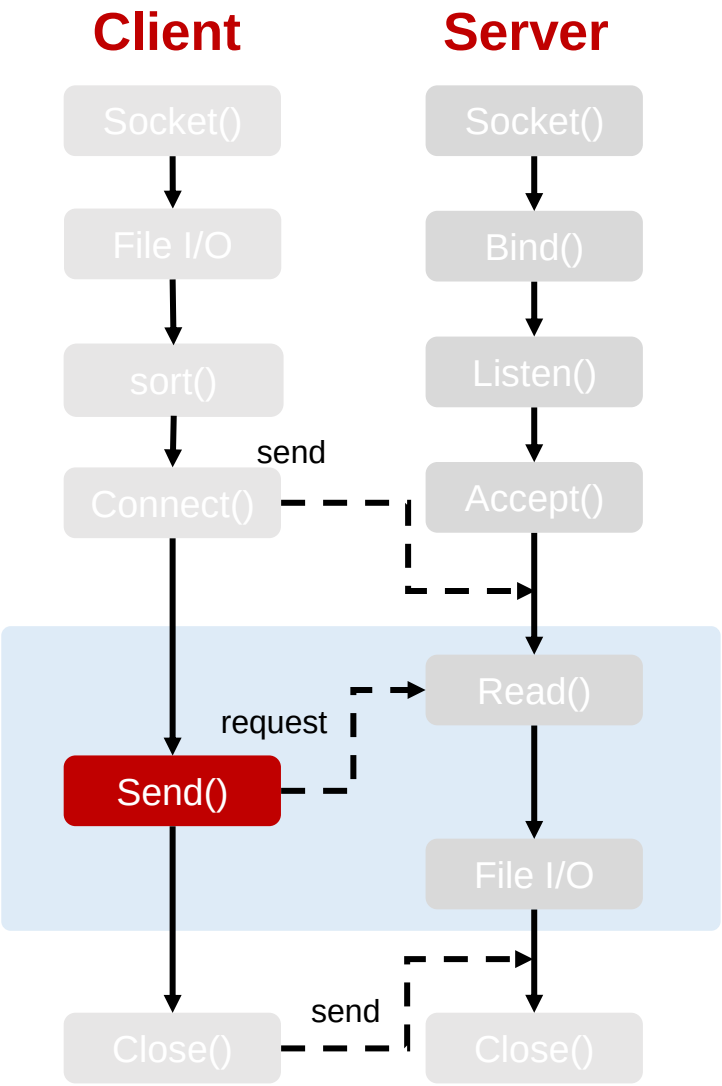
```
sendto(sockfd, (char*)Transcripts, 10 * sizeof(score), 0, (const struct sockaddr*)&servAddr, sizeof(servAddr));
```

将结构体数组指针强制转换成字符指针（其实就是字符串）

这种写法也对

```
sizeof(Transcripts)
```

send函数只接受char*类型的参数，这里的强制转换是为了骗过编译器，转换后实际的二进制数据不会发生改变，改变的只是编译器的解读方式，send函数发送的实际上是二进制数据



作业：Client.c

```
sendto(sockfd, (char*)Transcripts, 10 * sizeof(score), 0, (const struct sockaddr*)&servAddr, sizeof(servAddr));
```

这么写是错的 `strlen((char*)Transcripts)`

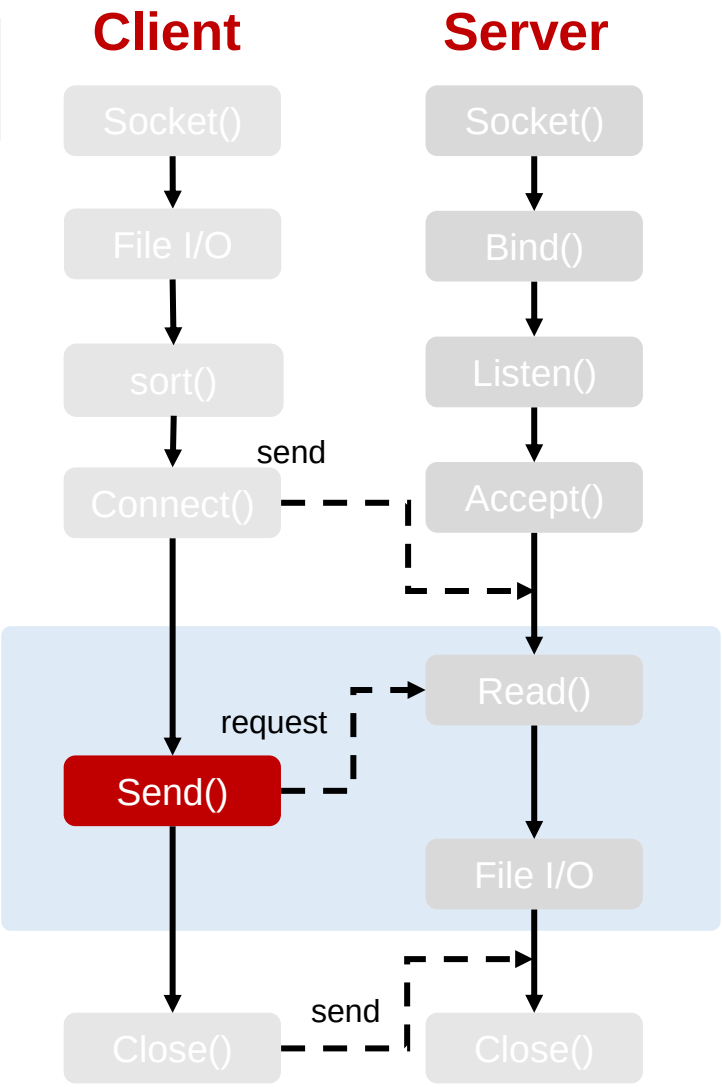
```
sendto(sockfd, (char*)Transcripts, 10 * sizeof(score), 0, (const struct sockaddr*)&servAddr, sizeof(servAddr));

char* view_ = (char*)Transcripts;
char view[280];
for (int i = 0; i < 10 * sizeof(score); i++)
    view[i] = view_[i];
//sendto(sockfd, szText, strlen(szText), 0, (const struct sockaddr*)&servAddr, sizeof(servAddr));

// 关闭套节字
closesocket(sockfd);
return 0;
}
```

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
view	0x000000657c92f330 "\a"	char[280
[0]	7 '\a'	char
[1]	0 '\0'	char
[2]	0 '\0'	char
[3]	0 '\0'	char
[4]	68 'D'	char
[5]	97 'a'	char
[6]	118 'v'	char
[7]	105 'i'	char
[8]	100 'd'	char
[9]	0 '\0'	char
[10]	0 '\0'	char
[11]	0 '\0'	char
[12]	0 '\0'	char

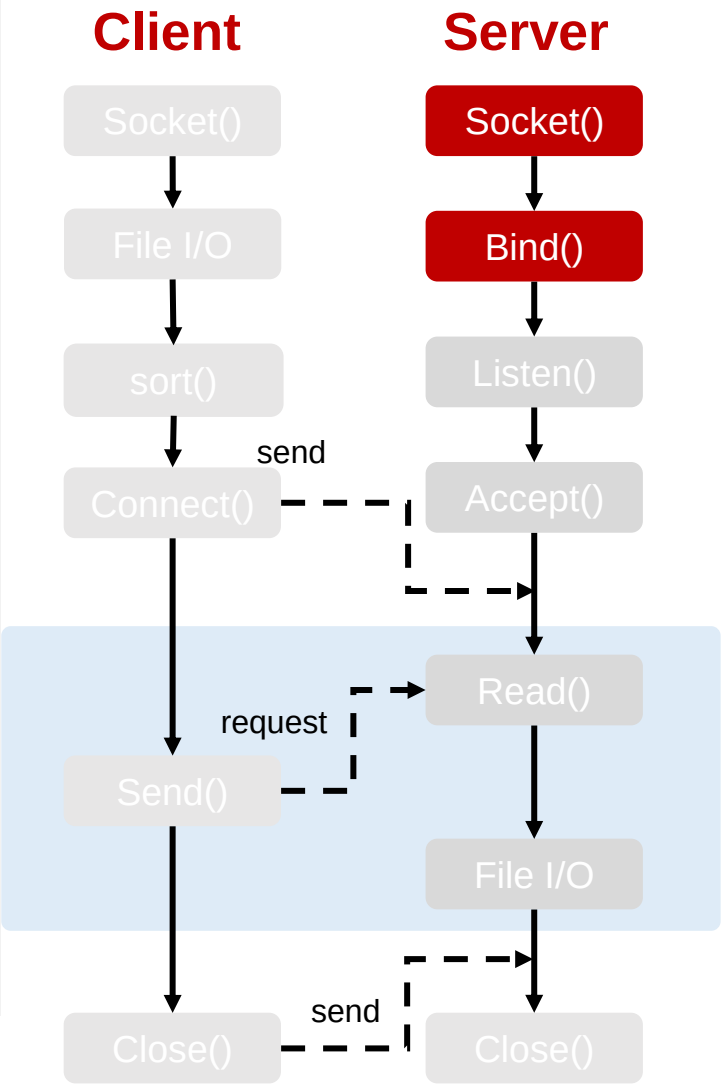
转换成char*后的Transcripts里面有很多\0，这是结构体内存对齐造成的，感兴趣的同学可以看发的链接



作业：Server.c

```
CInitSock_func(2, 2);

SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sockfd == INVALID_SOCKET)
{
    printf("Failed socket() \n");
    return 0;
}


struct sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(4567);
sin.sin_addr.S_un.S_addr = INADDR_ANY;

int flag = bind(sockfd, (const struct sockaddr*)&sin, sizeof(sin));

if (flag == SOCKET_ERROR)
{
    printf("Failed bind() \n");
    return 0;
}
```

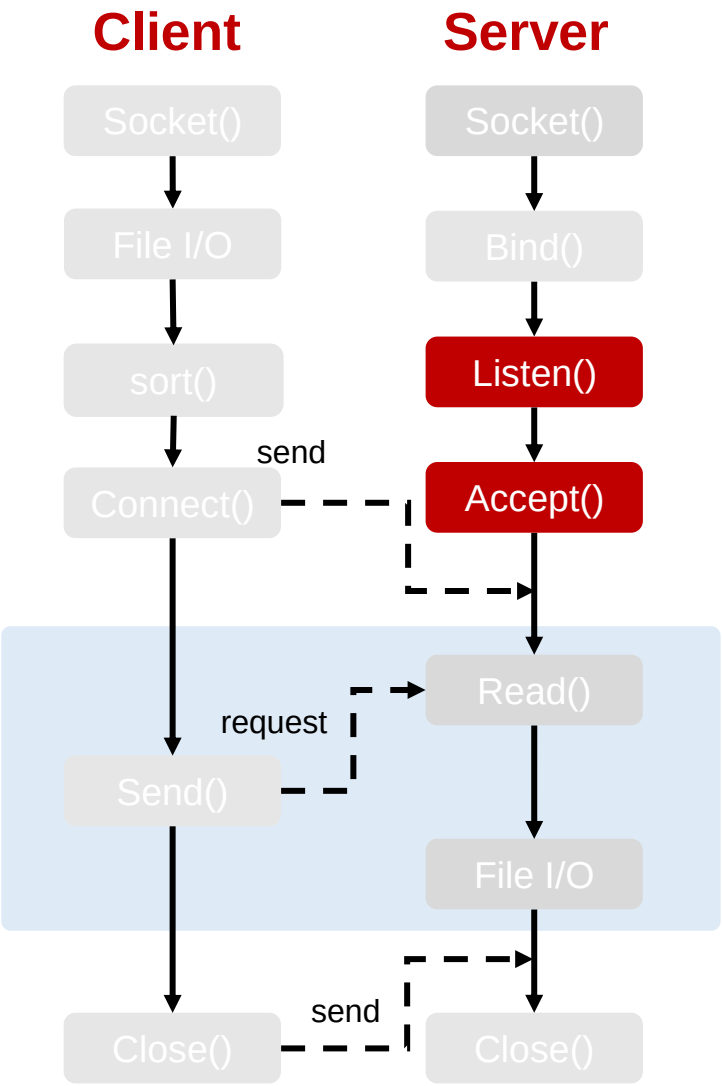
用的时候照抄就行，
用法固定



作业：Server.c

```
flag = listen(sockfd, 2);
if (flag == SOCKET_ERROR)
{
     printf("Failed listen() \n");
     return 0;
}

struct sockaddr_in remoteAddr;
int nAddrLen = sizeof(remoteAddr);
SOCKET sClient = 0;

sClient = accept(sockfd, (struct sockaddr*) & remoteAddr, &nAddrLen);
if (sClient == INVALID_SOCKET)
{
          printf("Failed accept()");
}
printf("接受到一个连接: %s \r\n", inet_ntoa(remoteAddr.sin_addr));
```

用的时候照抄就行，
用法固定

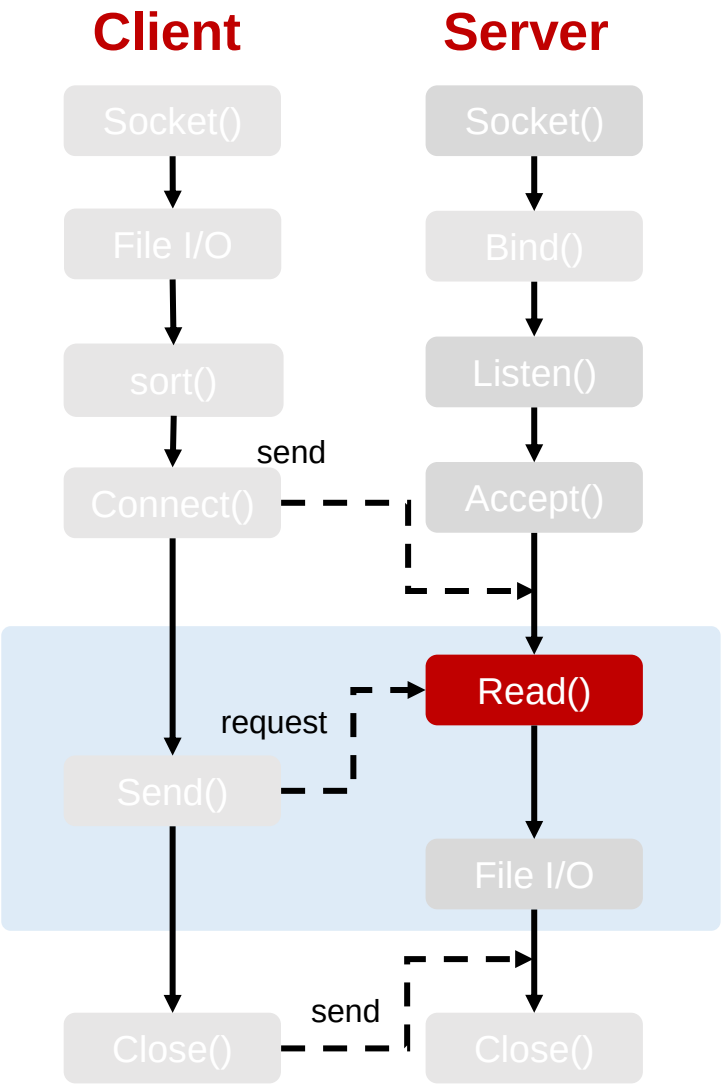


作业：Server.c

转换成char*来骗过编译器

这里参考client.c

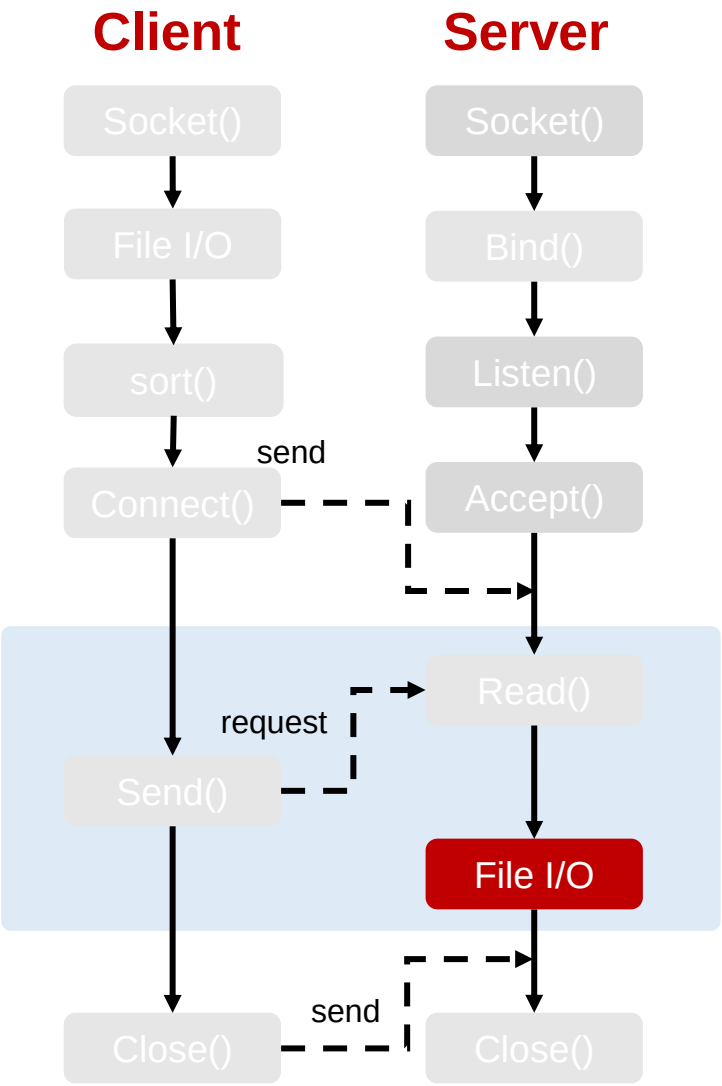
```
score Transcripts[10];
int nRecv = recv(sClient, (char*)Transcripts, 10 * sizeof(score), 0);
if (nRecv > 0)
{
        for (int i = 0; i < 10; i++)
        {
                printf("%d,%s,%d\n", Transcripts[i].stu_number,
                Transcripts[i].name,     Transcripts[i].score);
        }
}
```



作业：Server.c

这里如果不写绝对路径，文件不好找

```
FILE* fptr;
fptr = fopen("C://Users//ydf19//Desktop//data.csv", "w");

for (int i = 0; i < 10; i++)
{
        fprintf(fptr,"%d,%s,%d\n",
        Transcripts[i].stu_number, Transcripts[i].name,
        Transcripts[i].score);
}

fclose(fptr);
```



Content

1. File I/O & socket I/O
2. **Head files**
3. Multi-threading (不做要求！)

What is head file?

A header file is **a file with extension .h,** which contains C function declarations and macro definitions that can be used by different source files.

This is c file → run.c

This is head file → stdio.h

What is head file?

Two types of head file

C compiler defined head file

stdio.h

User defined head file

myFun.h

Avoid using names of C compiler defined head files!

Why using head file?

When your program grows large, it’s impossible to keep all functions in one file.

You can move parts of a program (functions) to separate files, and link them by head file.

You already used C
Compile defined head file

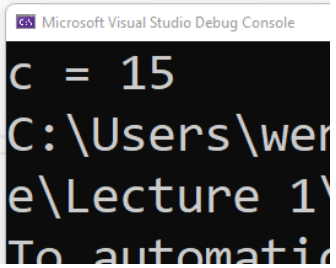
```
#include<stdio.h>

main()
{
    printf("Hello World!");
}
```

An example of function

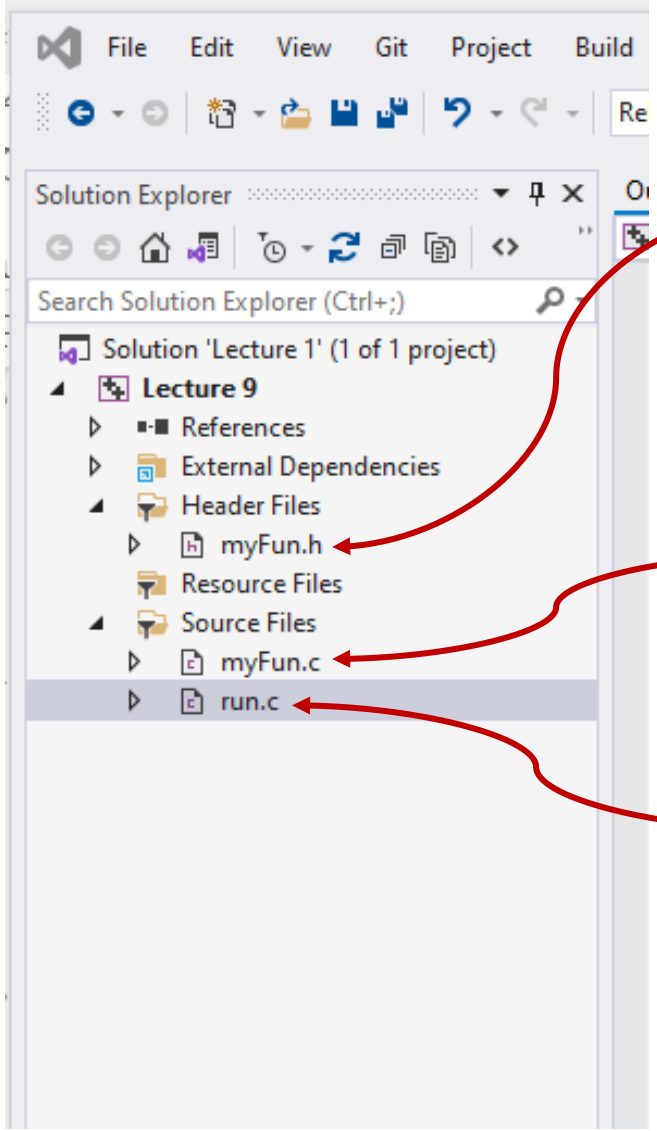
Self-defined function

- **Declaration**
- **Main function**
- **Definition**

```c
#include<stdio.h>

int sum(int x, int y);

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    printf("c = %d", c);
}

int sum(int x, int y)
{
    return x + y;
}
```



How to use head files?



1. 右键单击“Header Files”，添加一个新的头文件 myFun.h，声明方程

2. 右键单击“Source Files”，添加一个新的C文件 myFun.c，实现方程

3. 在run.c里添加代码 #include “myFun.c”，包含方程声明

How to use head files?

```c
#include<stdio.h>

int sum(int x, int y);

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    printf("c = %d", c);
}

int sum(int x, int y)
{
    return x + y;
}
```

Declaration

Main

Definition

myFun.h
(declaration)

```c
int sum(int x, int y);
```

run.c

```c
#include<stdio.h>
#include"myFun.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    printf("c = %d", c);
}
```

myFun.c
(definition)

```c
#include"myFun.h"

int sum(int x, int y)
{
    return x + y;
}
```

How to use head files?

Include the declaration of sum() by including myfun.h

```c
#include<stdio.h>
#include"myFun.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    printf("c = %d", c);
}
```

run.c

```c
int sum(int x, int y);
```

myFun.h
(declaration)

```c
#include"myFun.h"

int sum(int x, int y)
{
    return x + y;
}
```

myFun.c
(definition)

How to use head files?

Use **<>** to include
C compiled head file

```
#include<stdio.h>
```

Use **“”** to include
user defined head file

```
#include"myFun.h"
```

Use head file to declare more functions

```c
#include<stdio.h>

int sum(int x, int y);
int mul(int x, int y);

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}

int sum(int x, int y)
{
    return x + y;
}

int mul(int x, int y)
{
    return x * y;
}
```

Declaration

```c
int sum(int x, int y);
int mul(int x, int y);
```

myFun.h
(declaration)

Main

```c
#include<stdio.h>
#include"myFun.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}
```

run.c

Definition

```c
#include"myFun.h"

int sum(int x, int y)
{
    return x + y;
}

int mul(int x, int y)
{
    return x * y;
}
```

myFun.c
(definition)

Use head file to declare more functions

```c
#include<stdio.h>

int sum(int x, int y);
int mul(int x, int y);

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}

int sum(int x, int y)
{
    return x + y;
}

int mul(int x, int y)
{
    return x * y;
}
```

Declaration

myFun1.h

```c
int sum(int x, int y);
```

myFun2.h

```c
int mul(int x, int y);
```

Main

run.c

```c
#include<stdio.h>
#include"myFun1.h"
#include"myFun2.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}
```

Definition

myFun1.c

```c
#include"myFun1.h"

int sum(int x, int y)
{
    return x + y;
}
```

myFun2.c

```c
#include"myFun2.h"

int mul(int x, int y)
{
    return x * y;
}
```

Use head file to declare more functions

myFun1.h

```c
int sum(int x, int y);
```

myFun1.c

```c
#include"myFun1.h"

int sum(int x, int y)
{
    return x + y;
}
```

Library 1

myFun2.h

```c
int mul(int x, int y);
```

myFun2.c

```c
#include"myFun2.h"

int mul(int x, int y)
{
    return x * y;
}
```

Library 2

run.c

```c
#include<stdio.h>
#include"myFun1.h"
#include"myFun2.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}
```

Main function

Use head file to declare more functions

```c
#include<stdio.h>

int sum(int x, int y);
int mul(int x, int y);

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}

int sum(int x, int y)
{
    return x + y;
}

int mul(int x, int y)
{
    return sum(x + y)*y;
}
```

Declaration

myFun1.h

```c
int sum(int x, int y);
```

myFun2.h

```c
int mul(int x, int y);
```

Main

run.c

```c
#include<stdio.h>
#include"myFun.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}
```

Definition

myFun1.c

```c
#include"myFun1.h"

int sum(int x, int y)
{
    return x + y;
}
```

myFun2.c

```c
#include"myFun1.h"
#include"myFun2.h"

int mul(int x, int y)
{
    return sum(x + y)*y;
}
```

Use head file to declare more functions

myFun1.h

```c
int sum(int x, int y);
```

myFun1.c

```c
#include"myFun1.h"

int sum(int x, int y)
{
    return x + y;
}
```

Library 1

myFun2.h

```c
int mul(int x, int y);
```

myFun2.c

```c
#include"myFun1.h"
#include"myFun2.h"

int mul(int x, int y)
{
    return sum(x, y) * y;
}
```

Library 2

run.c

```c
#include<stdio.h>
#include"myFun1.h"
#include"myFun2.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
    int d = mul(a, b);
}
```

Main function

dependency

Step-by-step to use head files

① write head file **xxx.h** (declare functions)

```c
int sum(int x, int y);
```

myFun.h

② write c file **xxx.c** (include xxx.h, define functions)

```c
#include"myFun.h"

int sum(int x, int y)
{
    return x + y;
}
```

myFun.c

③ write **run.c** (include xxx.h, call functions)

```c
#include<stdio.h>
#include"myFun.h"

main()
{
    int a = 10, b = 5;
    int c = sum(a, b);
}
```

run.c

Case study of using head file

run.c

```c
#include <stdio.h>
#include "display.h"
#include "sort.h"

int main()
{
   score Transcripts[10];
   FILE* fptr;
   errno_t errw;
   errw = fopen_s(&fptr, "C:\\data.bin", "rb");
   fread(Transcripts, sizeof(score), 10, fptr);
   insertion_sort(Transcripts, 10);
   display_trans(Transcripts, 10);
}
```

display.c

```c
#include <stdio.h>
#include "display.h"

void display_trans(score * ptr, int length){
   for (int i = 0; i < length; i++) {
   printf("%d,%s,%d\n", ptr[i].stu_number,
ptr[i].name,ptr[i].score);
                          }
}
```

display.h

```c
typedef struct
{
   int stu_number;
   char name[20];
   int score;
}score;
void display_trans(score* ptr, int length);
```

sort.c

```c
#include <stdio.h>
#include "display.h"

void insertion_sort(score* ptr, int length) {
   for (int i = 1; i < length; i++)
   {
     for (int j = i; j > 0; j--) {
        if (ptr[j].score > ptr[j - 1].score)
   {
            score tempp = ptr[j];
            ptr[j] = ptr[j - 1];
            ptr[j - 1] = tempp;
                                  }
                        }
          }
}
```

sort.h

```c
void insertion_sort(score* ptr, int length);
```

```
7,David,99
3,Steve,98
5,Emma,97
2,Tim,95
4,Mara,94
1,Jack,93
8,Tyler,92
6,Paula,91
9,Noah,89
10,Daniel,85
```

Split the struct sorting function into different files

Case study of using head file

Write a function that put elements in the array in the reverse order and return the pointer of the array (e.g. 1,2,3,4,5,6 -> 6,5,4,3,2,1)

test.h

```c
#include <stdio.h>
void swap(int* a, int* b);
void inv_array(int* a, int size);
```

run.c

```c
#include "test.h"
main()
{
    int array[] =
    { 0,10,20,30,40,50,60,70 };
    inv_array(array, 8);
    for (int i = 0; i < 8; i++)
        printf("%d ", array[i]);
}
```

test.c

```c
#include "test.h"
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void inv_array(int* a, int size)
{
    for (int i = 0; i < size / 2; i++)
        swap(&a[i], &a[size - i - 1]);
}
```

Microsoft Visual Studio 调试控制台

```
70 60 50 40 30 20 10 0
```

Content

1. File I/O & socket I/O
2. Head files
3. **Multi-threading (不做要求！)**

Multi-threads in our life

Multi-threads: do different things at the same time!

Multi-threads in our life

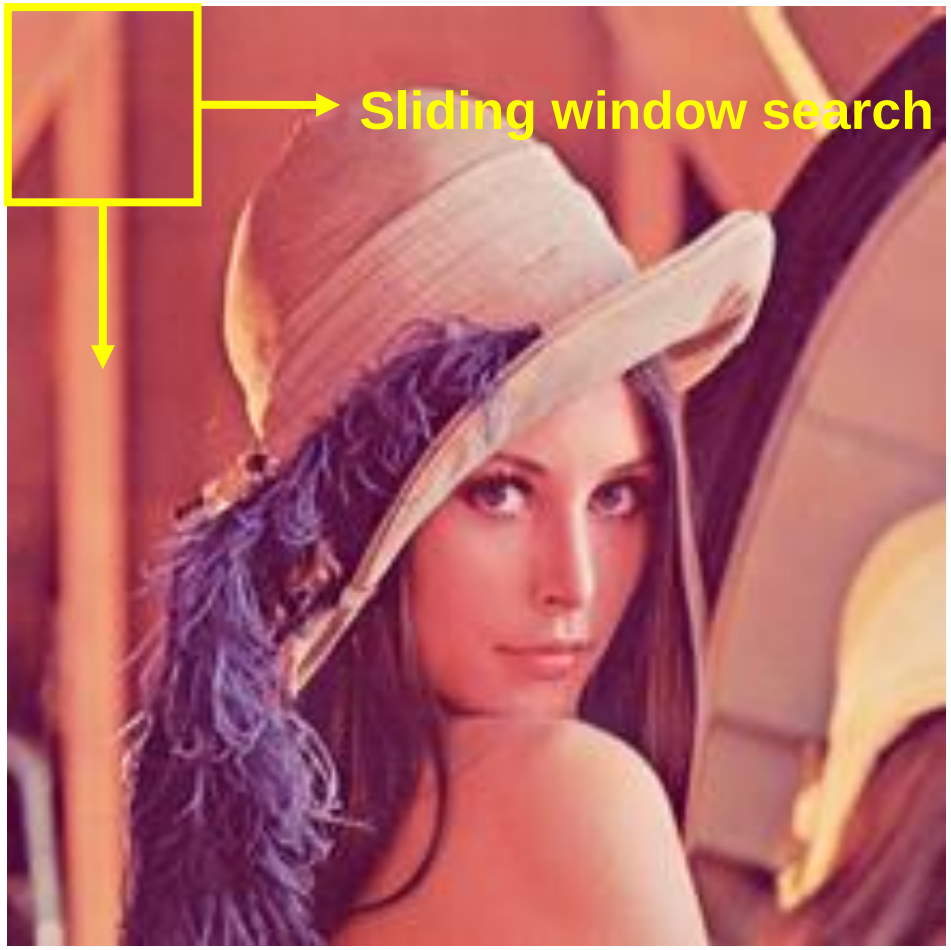
Running while listening music

Running while watching movie

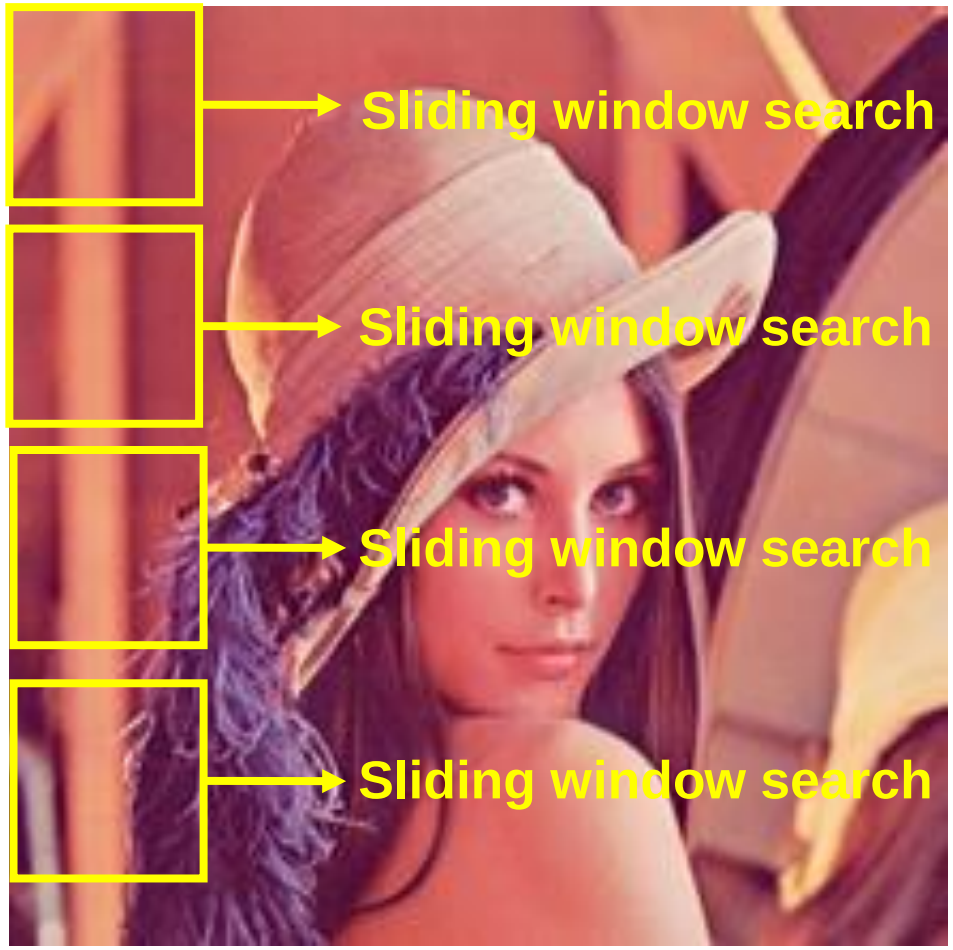
Multi-threads in our life

Singing while dancing!

Why multi-threads is needed?



Efficiency!



Process and thread

Process (进程)

程序运行时系统会分配内存资源

Thread (线程)

线程是进程在运行时CPU的调度单位

Process and thread

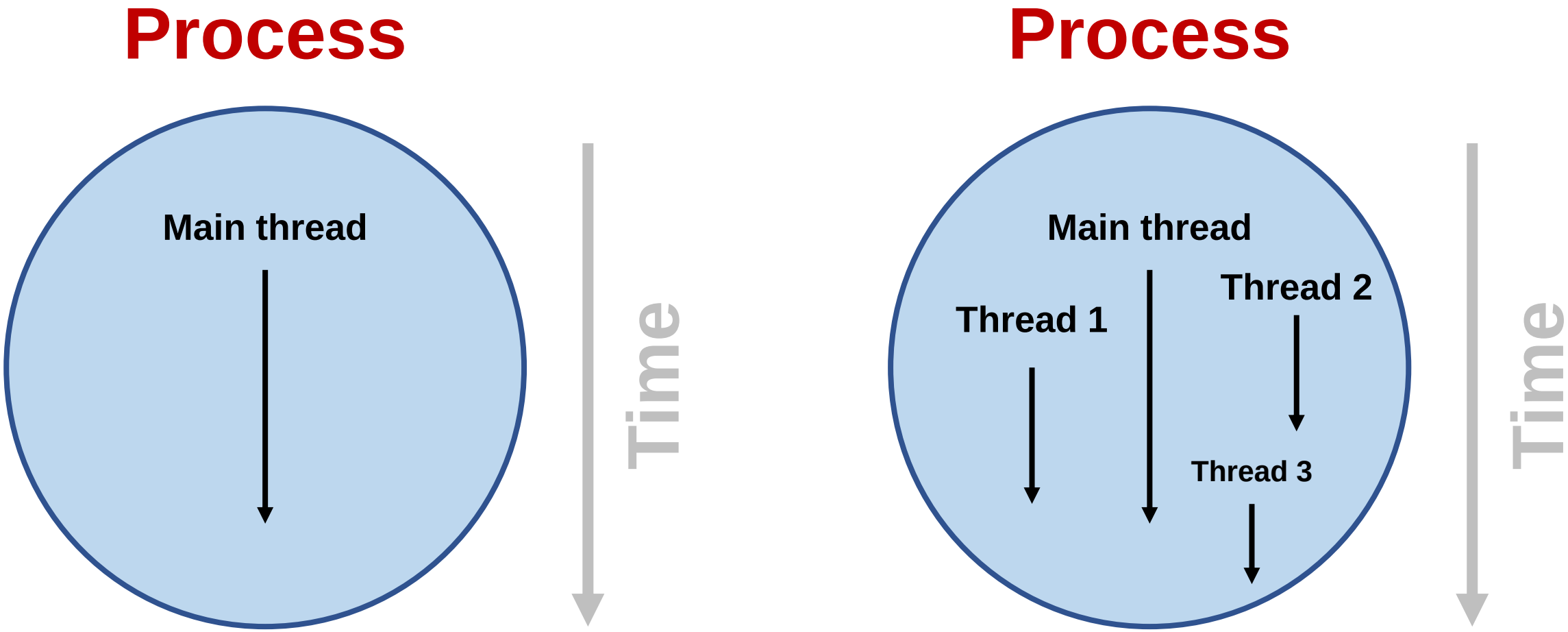
Process (进程)

- ✓ Independent of each other
- ✓ Separate address spaces

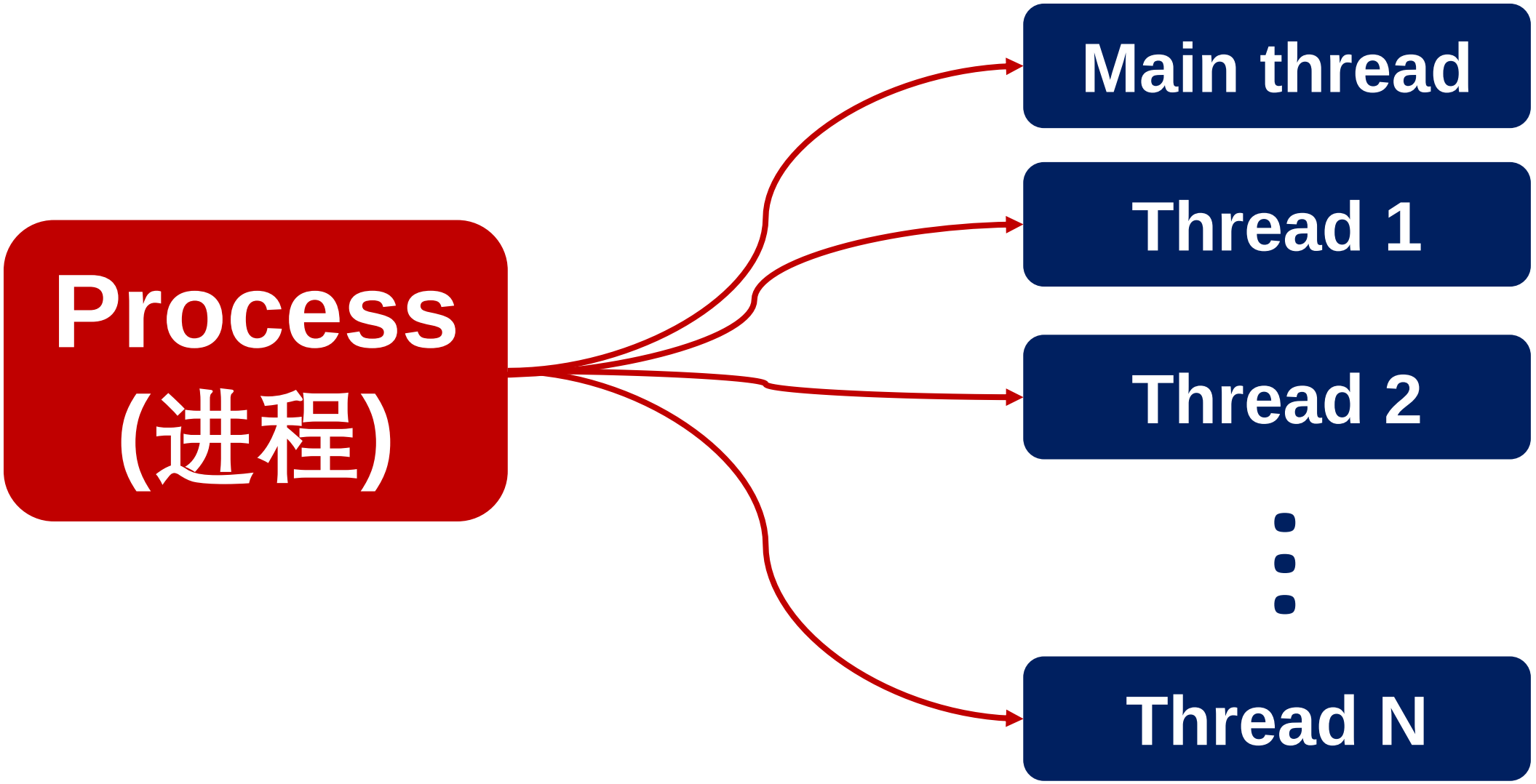
Thread (线程)

- ✓ Subsets of a process
- ✓ Multiple threads in a process
- ✓ Share address space in a process

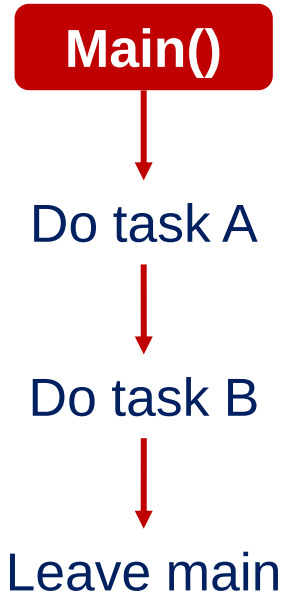
Process and thread

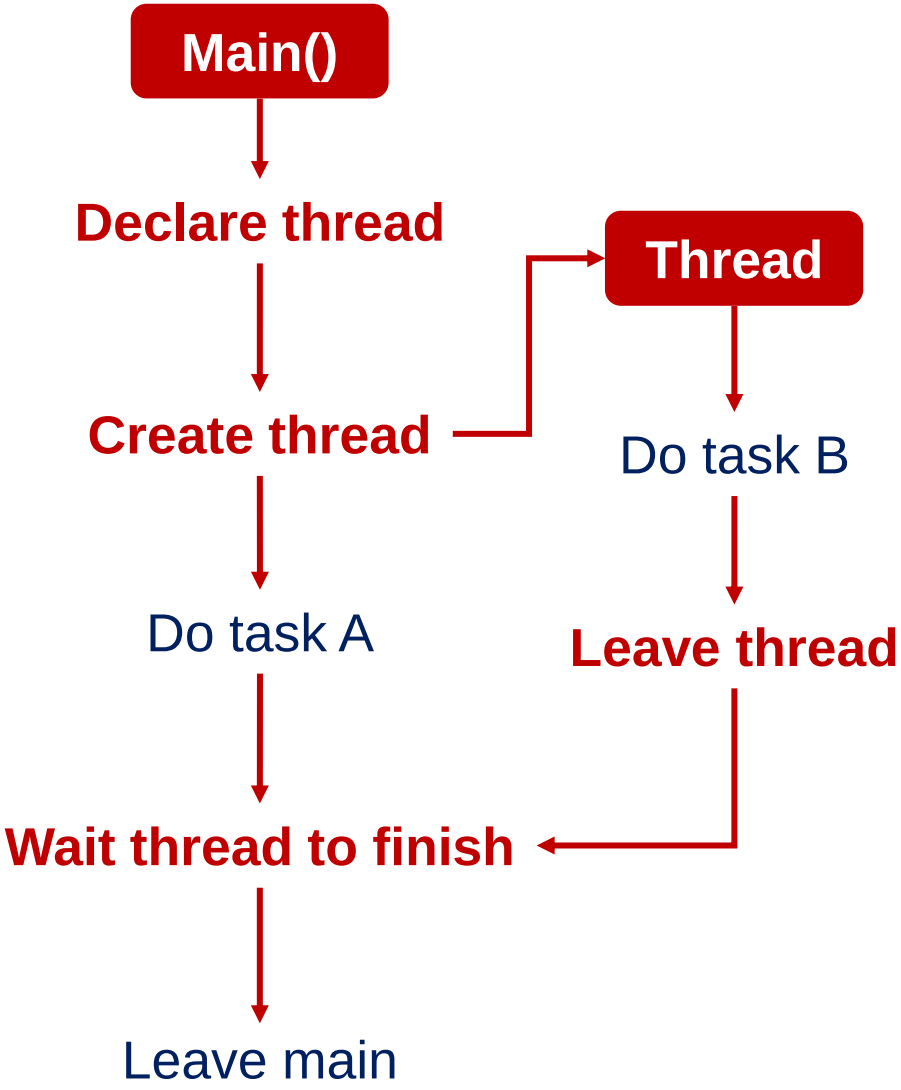


Process and thread

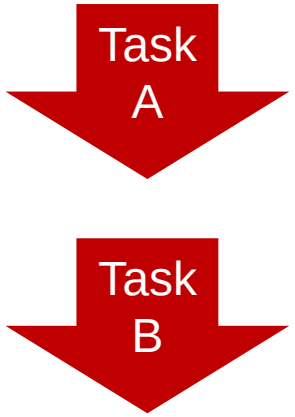


Single thread VS multi-threads





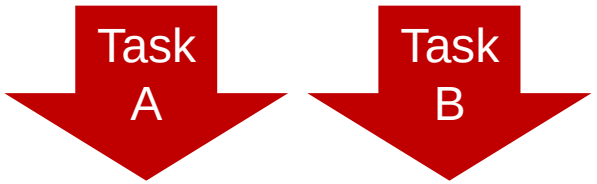
Single thread VS multi-threads



sequential

```c
#include<stdio.h>

int main()
{
    // do task A

    // do task B

    return 0;
}
```

```c
#include<stdio.h>
#include<pthread.h>

void *perform_task()
{
    // do task B

    thread_exit(NULL);
}

int main()
{
    // declare a thread
    pthread_t thread;

    // start the thread
    pthread_create(&thread, NULL, perform_task, NULL);

    // do task A

    // wait task B to finish
    thread_join(thread, NULL);

    return 0;
}
```



simultaneous

Step-by-step to use thread

① declare a thread

```
pthread_t thread;
```

② create/start the thread

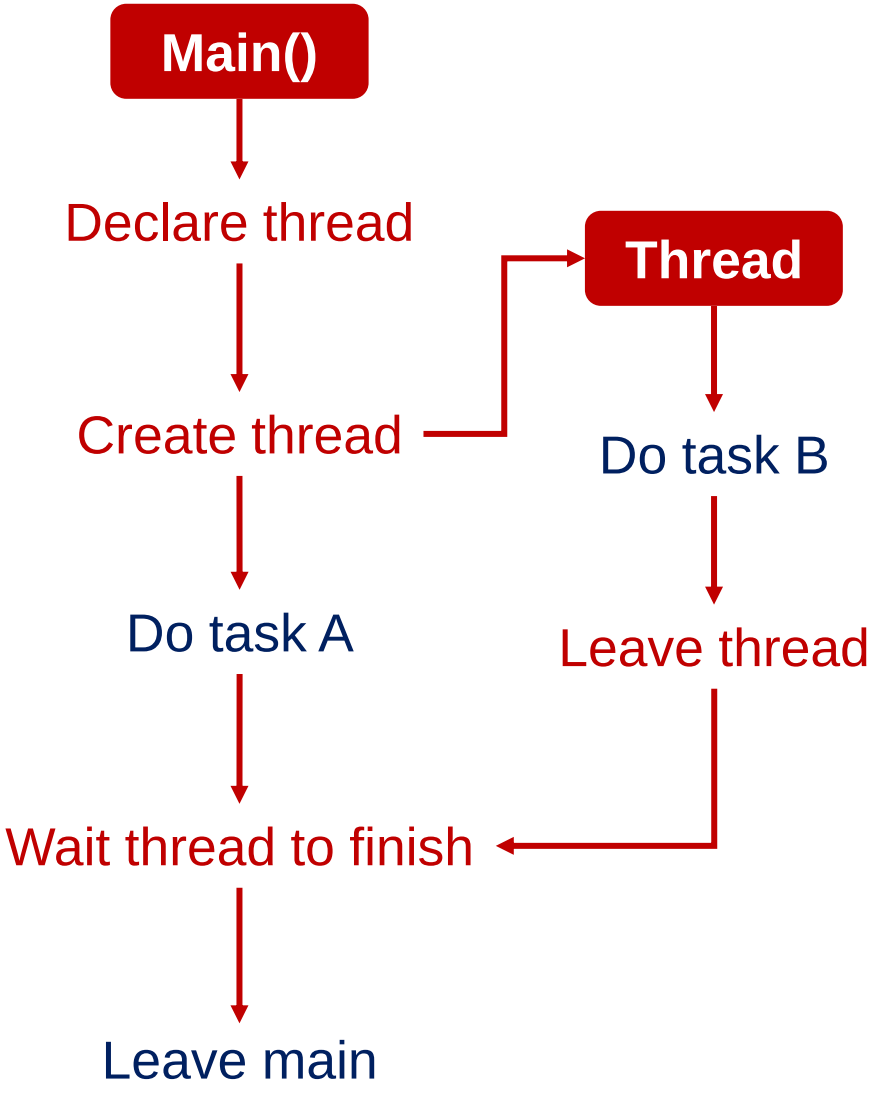
```
int pthread_create(pthread_t *thread, const pthread_attr_t
*attr, void * (*start_routine)(void *), void *arg);
```

② leave the thread

```
void pthread_exit (void *retval)
```

③ wait the thread to finish (main)

```
int pthread_join( pthread_t thread , void ** value_ptr );
```



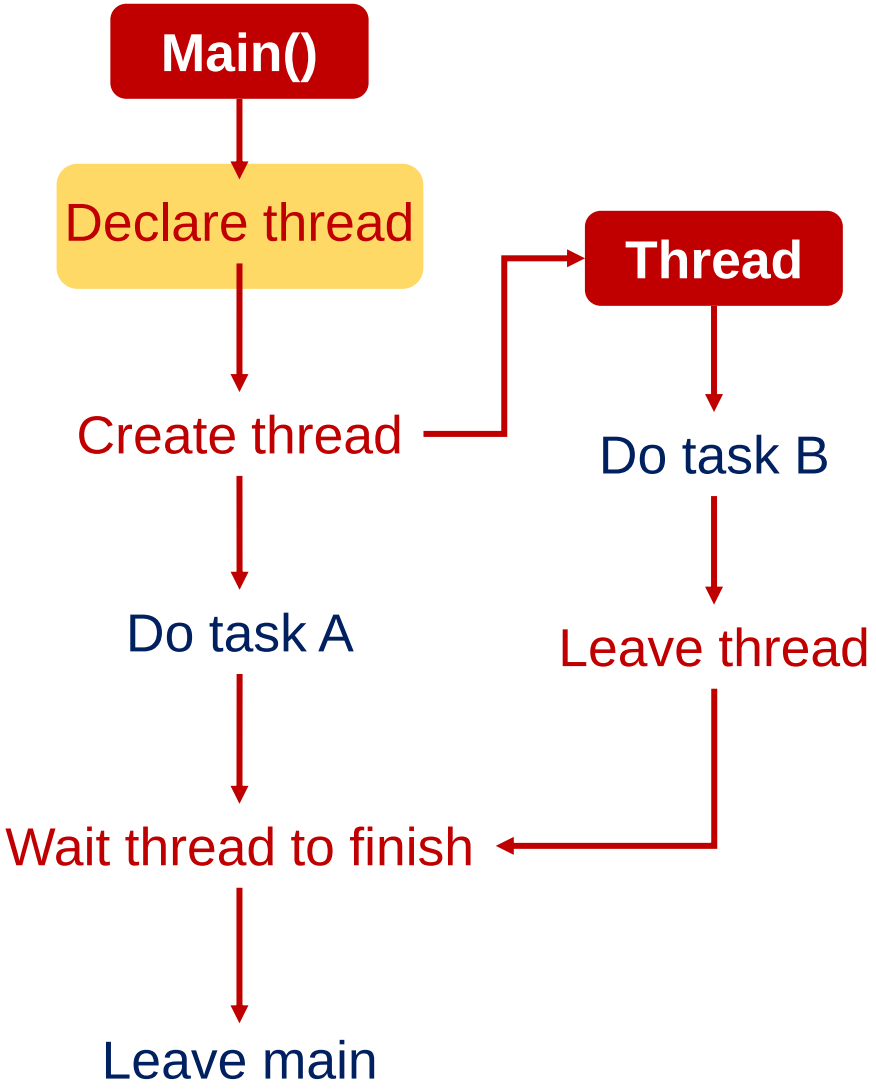
Step 1: declare a thread

Pthread is a C compiler function, include its head file

```
#include<pthread.h>
```

```
pthread_t thread;
```

声明了一个线程标识符ID



Step 2: create/start the thread

Flag
成功：0
失败：-1

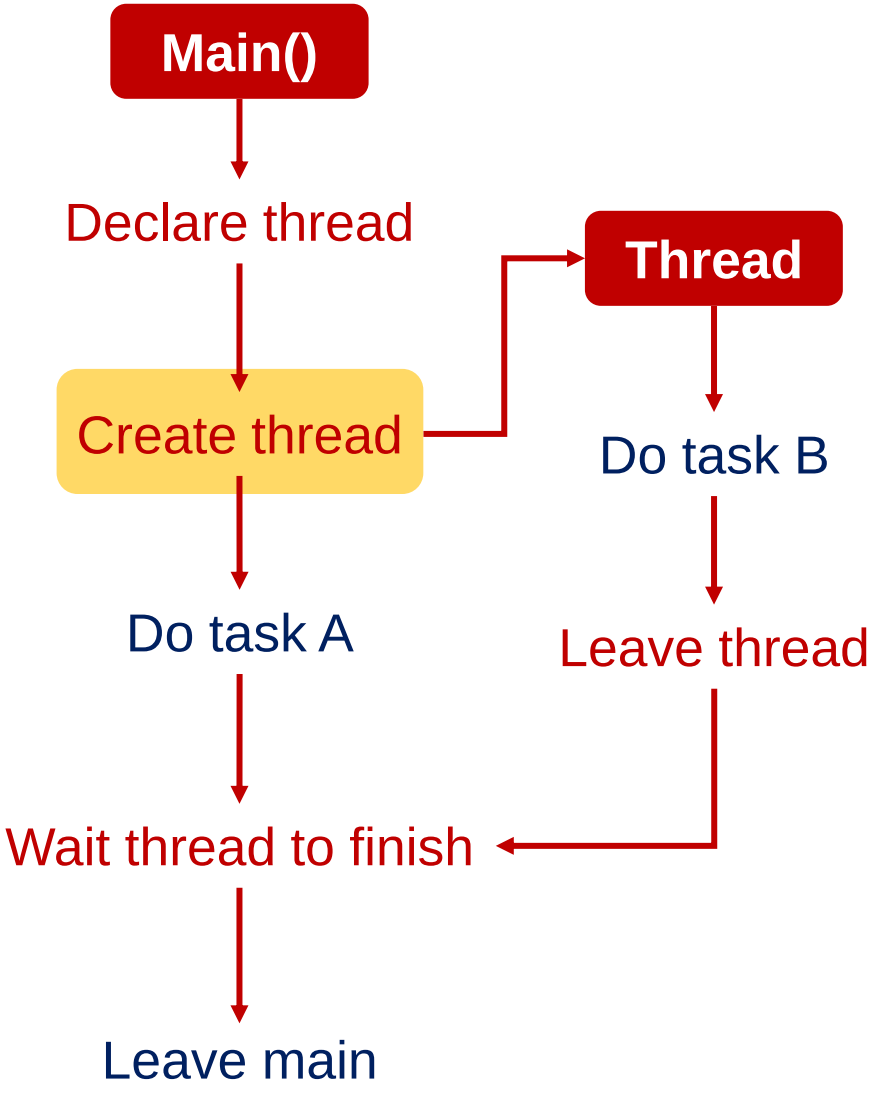
放入声明的线程
标识符ID

```
int pthread_create(pthread_t * thread, const
pthread_attr_t * attr, void*(*start_routine)(void*),
void* arg);
```

可以设定线程属性（优先级等），没有用NULL

线程运行的函数指针（地址）

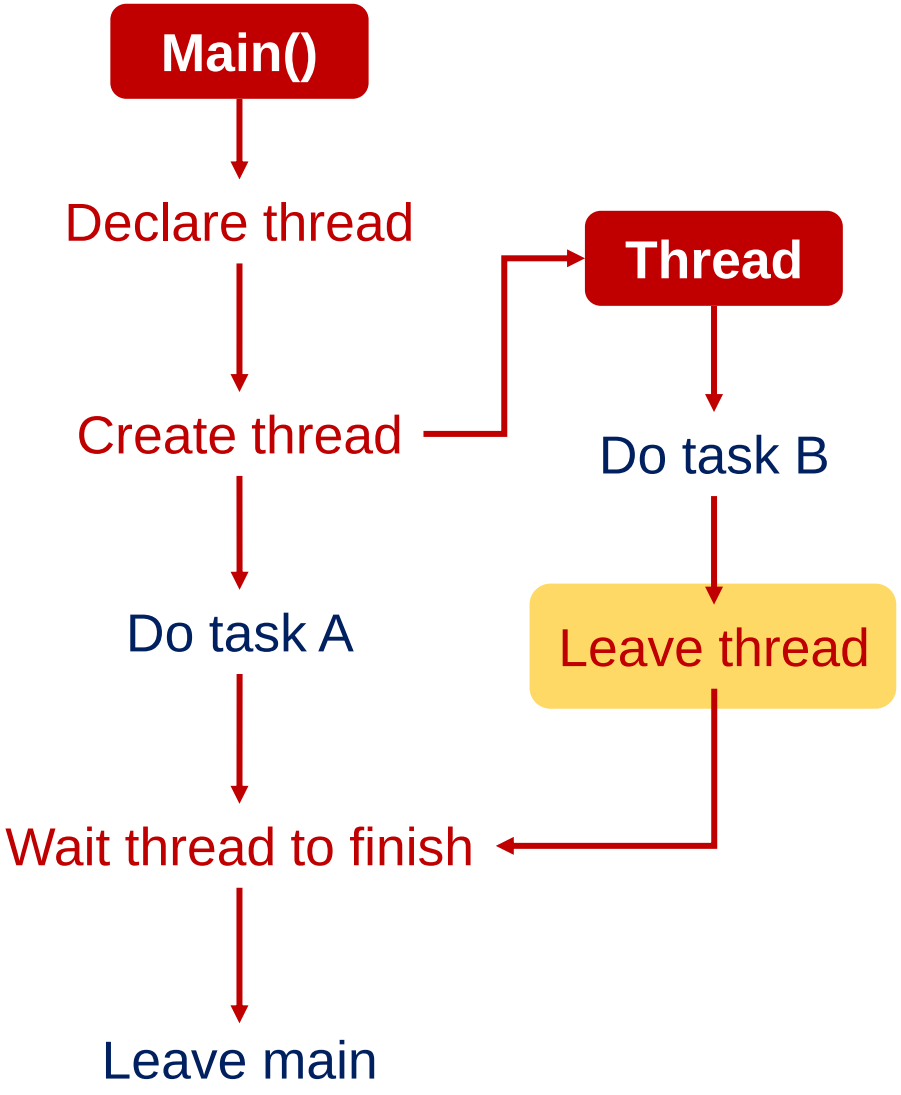
向线程输入的参数，
没有用NULL



Step 3: leave the thread

```
void pthread_exit(void* value_ptr);
```

线程运行结束，退出时
要返回的参数

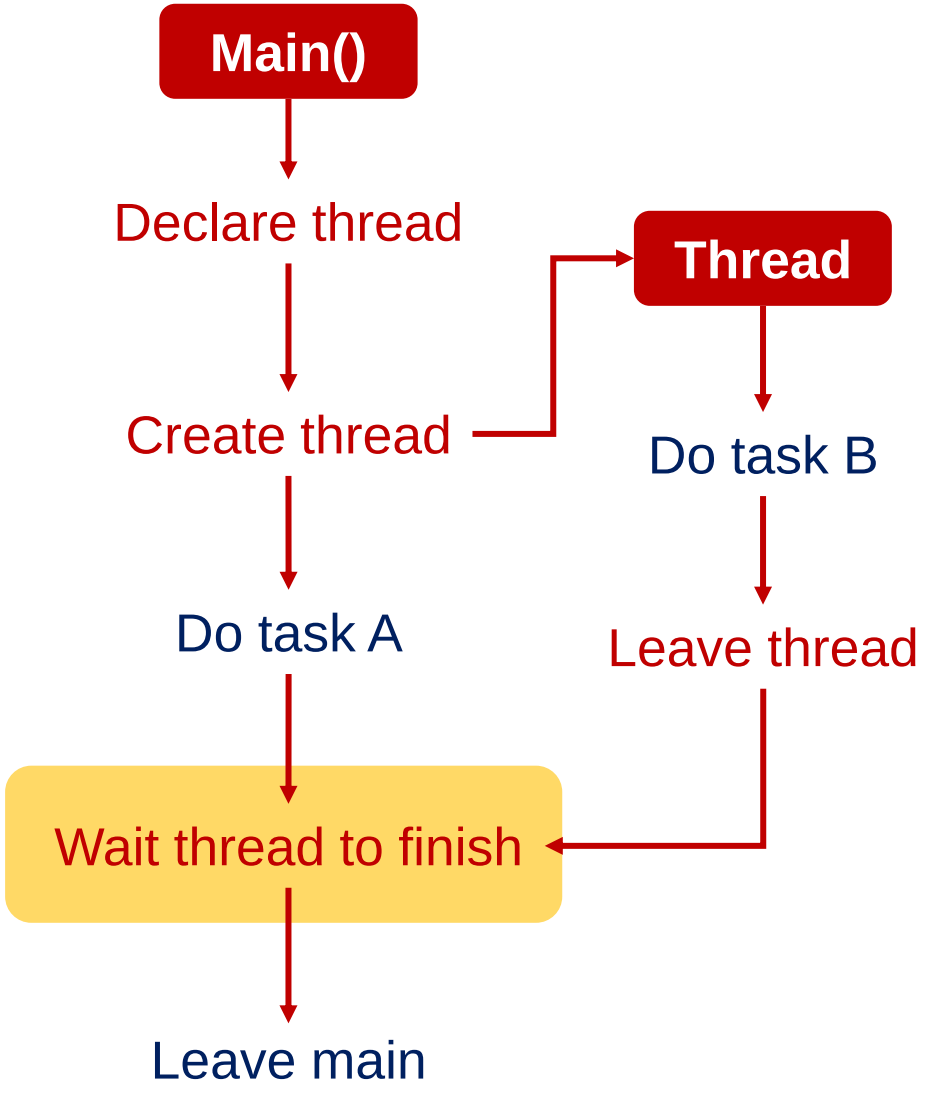


Step 4: wait thread to finish

放入声明的线程标识符ID

```
int pthread_join(pthread_t thread,
void** value_ptr);
```

主函数获取线程输出的参数



Case study: say hello & bye

Single thread (先说Hello再说Bye)

```c
#include<stdio.h>

int main()
{
    for (int i = 0; i < 10; i++)
        printf("Hello!\n");

    for (int i = 0; i < 10; i++)
        printf("Bye!\n");

    return 0;
}
```

```
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
```

Multiple threads (边说Hello边说Bye)

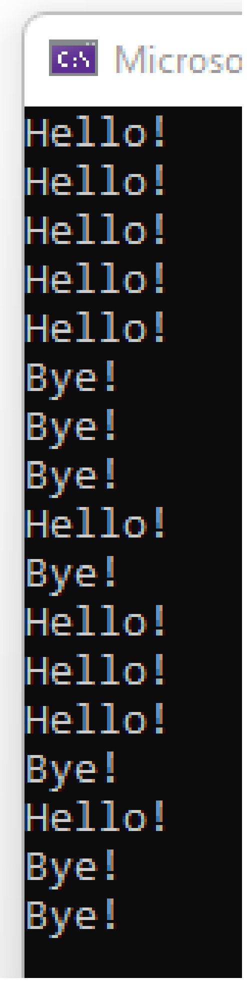
```c
#include<pthread.h>
#include<stdio.h>
#pragma comment(lib, "pthreadVC2.lib")

void* perform_task()
{
    for (int i = 0; i < 10; i++)
        printf("Bye!\n");

    pthread_exit(NULL);
}

int main()
{
    // declare a thread
    pthread_t thread;

    // start the thread
    pthread_create(&thread, NULL, perform_task, NULL);

    for (int i = 0; i < 10; i++)
        printf("Hello!\n");

    // wait task B to finish
    pthread_join(thread, NULL);

    return 0;
}
```

```
Hello!
Hello!
Hello!
Bye!
Bye!
Bye!
Hello!
Hello!
Hello!
Hello!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Hello!
Hello!
Bye!
Hello!
```

Case study: say hello & bye

说10次Hello，说100次Bye

```c
#include<pthread.h>
#include<stdio.h>
#pragma comment(lib, "pthreadVC2.lib")

void* perform_task()
{
    for (int i = 0; i < 100; i++)
        printf("Bye!\n");

    pthread_exit(NULL);
}

int main()
{
    // declare a thread
    pthread_t thread;

    // start the thread
    pthread_create(&thread, NULL, perform_task, NULL);

    for (int i = 0; i < 10; i++)
        printf("Hello!\n");

    // wait task B to finish
    //pthread_join(thread, NULL);

    return 0;
}
```

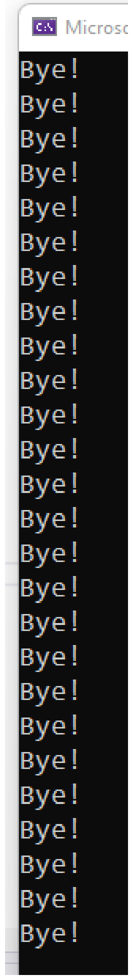


```
Hello!
Hello!
Hello!
Hello!
Hello!
Bye!
Bye!
Bye!
Hello!
Bye!
Hello!
Hello!
Hello!
Bye!
Hello!
Bye!
Bye!
```

说10次Hello，说100次Bye

```c
#include<pthread.h>
#include<stdio.h>
#pragma comment(lib, "pthreadVC2.lib")

void* perform_task()
{
    for (int i = 0; i < 100; i++)
        printf("Bye!\n");

    pthread_exit(NULL);
}

int main()
{
    // declare a thread
    pthread_t thread;

    // start the thread
    pthread_create(&thread, NULL, perform_task, NULL);

    for (int i = 0; i < 10; i++)
        printf("Hello!\n");

    // wait task B to finish
    pthread_join(thread, NULL);

    return 0;
}
```

阻塞！



```
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
Bye!
```

Case study: say hello & bye

代入参数，用5个线程对5个人说3次Hello！

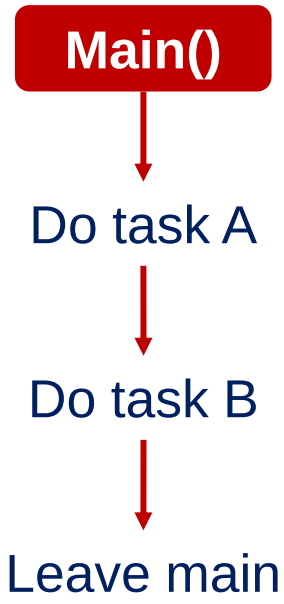
```c
#include<pthread.h>
#include<stdio.h>
#pragma comment(lib, "pthreadVC2.lib")

void* perform_task(void * param)
{
    for (int i = 0; i < 3; i++)
        printf("Hello %d!\n", (int*) param);

    pthread_exit(NULL);
}

int main()
{
    // declare threads
    pthread_t thread[5];

    // start threads
    for (int i = 0; i < sizeof(thread)/sizeof(thread[0]); i++)
    {
        pthread_create(&thread[i], NULL, perform_task, i);
    }
    // wait task B to finish
    for (int i = 0; i < sizeof(thread) / sizeof(thread[0]); i++)
    {
        pthread_join(thread[i], NULL);
    }
    return 0;
}
```

Microsoft Visual Studi

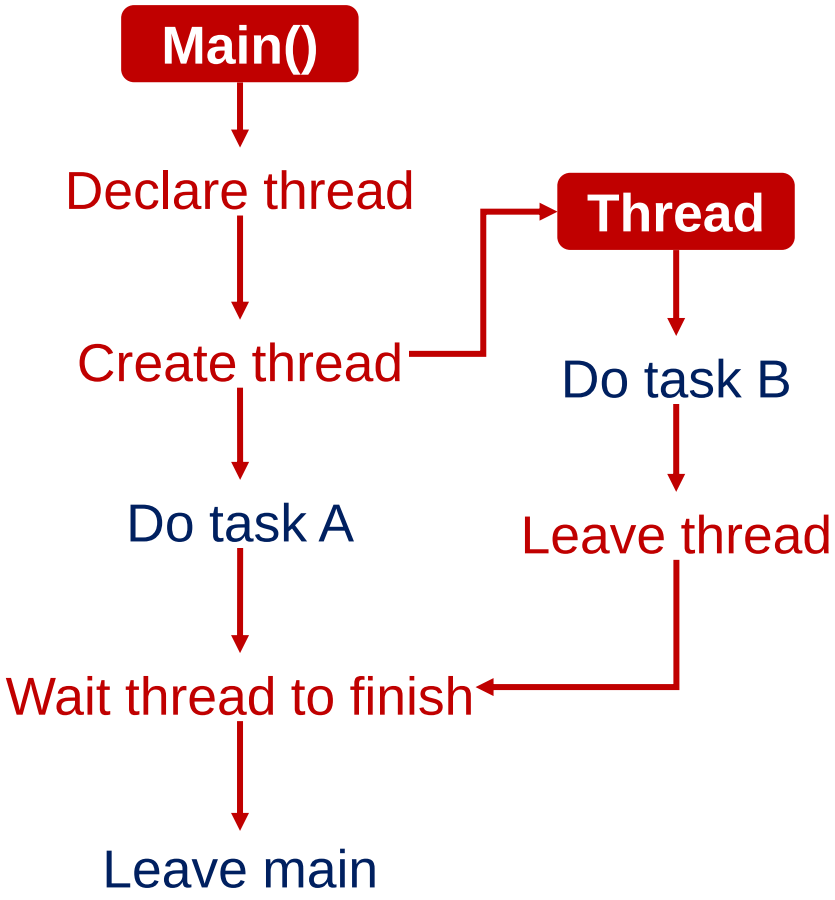
```
Hello 0!
Hello 0!
Hello 1!
Hello 0!
Hello 2!
Hello 2!
Hello 4!
Hello 4!
Hello 4!
Hello 3!
Hello 3!
Hello 3!
Hello 1!
Hello 1!
Hello 2!
```

Summary of multi-threads

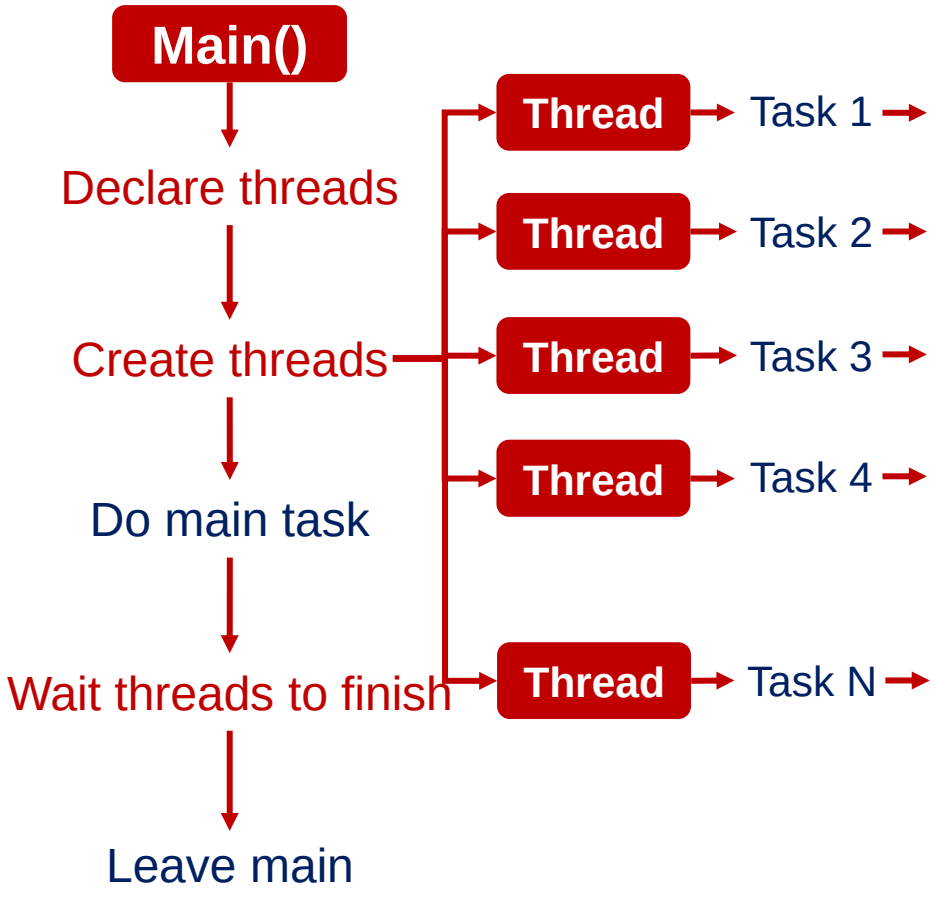
Single thread (main)



Two threads (main + thread)



Multiple threads (main + N threads)



Case study: two threads counting

```c
void* thread(void* a)
{
for (int i = 0; i < 30; i++)
{
printf("线程执行第 %d 次\n", i + 1);
}
}

void main()
{
pthread_t id;
int ret = pthread_create(&id, NULL, thread, NULL);
if (ret != 0)
{
printf("线程创建错误！\n");
}
for (int i = 0; i < 30; i++)
{
printf("main函数执行第 %d 次\n", i + 1);
}
pthread_join(id, NULL);
}
```

线程函数，你可以认为这是另一个main()函数，它会和main()函数同时执行

这个是线程的ID，作用和上节课的FILE* fptr差不多，是为了方便地找到这个线程进行操作

线程的参数，可为空

线程函数，可以将ID和线程函数关联起来

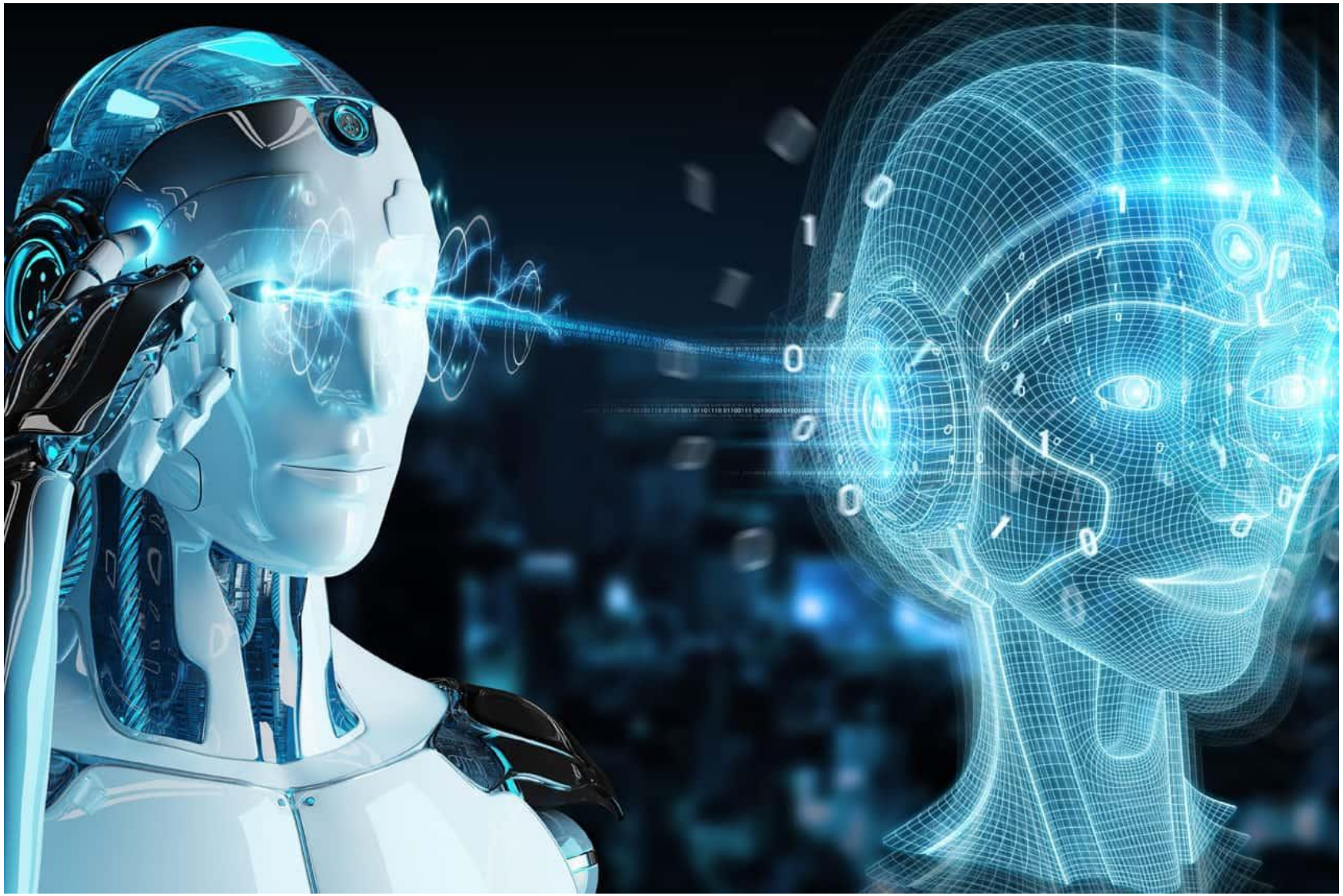
线程的属性，可为空

```
main函数执行第 1 次
main函数执行第 2 次
main函数执行第 3 次
main函数执行第 4 次
main函数执行第 5 次
main函数执行第 6 次
main函数执行第 7 次
main函数执行第 8 次
main函数执行第 9 次
线程执行第 1 次
线程执行第 2 次
线程执行第 3 次
线程执行第 4 次
main函数执行第 10 次
main函数执行第 11 次
main函数执行第 12 次
main函数执行第 13 次
main函数执行第 14 次
main函数执行第 15 次
线程执行第 5 次
线程执行第 6 次
线程执行第 7 次
main函数执行第 16 次
main函数执行第 17 次
main函数执行第 18 次
main函数执行第 19 次
main函数执行第 20 次
main函数执行第 21 次
main函数执行第 22 次
main函数执行第 23 次
```

Summary

- We have reviewed the **file I/O** and **socket I/O**
- You can use **head files** to split multiple functions, avoid writing big script with many different functions
- Typically, 1 .h file (declaration) and 1 .c file (definition) build up a library
- You know what is thread and basics of **multi-threading**. It is not demanded for this course, only need to know
- Time to strengthen the socket part (useful in the future).

Project II: intelligent agent



Project II: intelligent agent

Use server.c and a client.c to create an intelligent agent. The agent implemented on the server can use KNN (assignment of lecture 7) to answer query. There is a new point (coordinate) and we want to know which class it belongs to. We provide you a bin file that contains the coordinates and classes of 100 points which you can load on the server. You should do following:

1. On the client side, use scanf()/scanf_s() to enter a new point (coordinate) and the value of K, send new point and K value from client.c to server.c.

2. On the server side, use KNN to determine which class the new point belongs to, and send the class ID from server.c to client.c

3. On the client side, print the class ID received from server on the screen.

Project II: intelligent agent

a) Test input : (40, 65), K = 3; (40, 65), K= 7
b) The struct that contains the coordinates and class is shown below
c) You can download the bin file on bb

```
typedef enum category { A, B } Enum;

typedef struct point
{
    int x;
    int y;
    Enum category;
}Point;

point P[100];
```