网络编程技术

gcc编译

OSI，tcp/ip模型

应用---tcp数据流，传输----ip包头部，网络---以太网头部。。。。

<-----逆向

用户数据+app头部

ip数据报，一位当四个字节

06tcp，11udp 区别

网络通信双方五元组

URG紧急指针,ACK去确认序列号以及最大数据段的大小,SYN,PSH直接发到上层，不进缓冲区,RST拒绝,FIN请求关闭连接，关闭写可以收

通信连接，SYN=1（TCP）

connect

套接字编程

socket编程在应用层和传输层之间

通信协议

AF_LOCAL:un ix系统本地通信

AF_INET：IP4

AF_INET6:IP6

套接口类型宏定义

SOCKET_STREAM:TCP

SOCKET_DGRAM:UDP

SOCKET_RAW:原始

第一类地址结构

```
struct in_addr{
    in_addr_t s_addr;
}
```

Linux网络通信----htonl()、htons()、ntohl()、ntohs()四个函数_linux ntohs-CSDN博客

字节排序，小端或者大端

若字节序不一致，主机字节序需要通过函数转换成网络字节序

网络字节序为大端顺序，是相反的

IP v4地址转换函数

inet_addr

inet_aton

inet_ntoa,有结构体参数

ipv6

inet_pton

inet_ntop,注意给的长度需要能装下转换的

TCP套接字

套接字描述符相当于文件描述符，会记录下五元组

int socket(int family,int type,int protocol); 填相应的参数

int bind(int sockfd（客户端）,const struct sockaddr *addr（服务器）,socklen_len)

addr必须为网络字节序

getsockname

客户端可以调用，不过一般是服务器来分配ip和端口给客户端使用

bind失败原因：端口是否可以复用

setsockopt(fd,SOL_SOCKET,SO_REUSERADDR,&opt,sizeof(opt))

套接字调用函数过程

listen函数（只允许TCP，主动套接字转被动，等待客户端连接），建立队列条目排队（未完成-已完成）

int accept(int sockfd,struct sockaddr *cliaddr（服务器）,socklen_t *addrlen ）-1出错

取已完成队列，非负表示**返回套接字为已连接套接字**，与监听套接字区分开

已连接套接字产生一个关闭一个，监听套接字区一个到服务结束再关闭

close/shutdown

UDP套接字

recvfrom和sendto

进程到内核

内核到进程

并发服务器

迭代和并发服务器

父子进程拷贝：

fork（）调用一次，返回两次；子进程返回值0，父进程返回子进程的ID

父进程需要能找到自己的子进程

vfork（）：新老进程共享同样的资源，子先父阻

僵尸进程

wait（）和waitpid()

进程占据太多资源，所以有线程

pthread_join 联合线程，必须指定等待线程的ID，，和waitpid区分开

pthtread_detach 分离线程（create默认是联合的）

pthtread_exit或者main函数exit退出进程返回

pthread_self：获取自身id

pthread_once() 初始化函数放置位置

pthread_cancel()

线程编译-lpthread

pthread_mutex_lock：锁了其他要锁需等待

与多进程不一样的是，多线程是共享空间，过程中不能关闭监听和连接套接字

TSD线程特定数据使用，保证线程独立的一些好的用处

新增：

核心概念
通知（Signal）：
当一个线程修改了共享状态并满足某个条件时，可以通知等待的线程。
pthread_cond_signal：唤醒一个等待的线程。
pthread_cond_broadcast：唤醒所有等待的线程

期末考总结

填空

inet_addr（）将字符串形式的IP地址转换成32位的网络字节序的二进制IP地址

网络编程中的API是指 **应用程序编程接口**（Application Programming Interface），它定义了不同软件组件之间的交互方式，在网络编程中用于实现网络通信功能。

并发设计中的两种主要运行单元分别是：进程和线程

线程是**进程内的独立执行实体和调度单元**，一个进程内的所有线程共享相同的内存空间、全局变量等信息

socket()产生套接字。该函数调用必须给出所使用的**地址簇、套接字类型和协议标志。**该函数返回一个**套接字描述符。**

判断

2023-2024

1、Socket编程可支持对IP报头的自组包;√

通过**原始套接字（**`SOCK_RAW`**）可以自定义**IP报头（需权限，如`root`）。

2、UDP编程必须使用bind函数;x

UDP客户端通常不需要`bind`（由系统自动分配端口），但服务端需`bind`固定端口。

3、Socket函数中，IPv4和IPv4都是使用INET参数作为协议名;x

IPv4用`AF_INET`，IPv6用`AF_INET6`

4、若父进程要等待子线程结束，则子线程创建时不能设置detach 属性;√

`pthread_detach`会使线程结束后自动释放资源，无法被`pthread_join`等待。

5、 signal函数可以设定自定义的信号响应处理过程;√

6、 select函数可之作为定时函数使用;√
7、exec函数与 fork函数的执行行为是一致的;x

`fork`创建子进程（复制父进程），`exec`替换当前进程映像（不创建新进程）

8、一个完整的网络连接应包含本地IP,本地Port,以及对等端P,对等端Port的信息√

9、可以用socket编程方法或者修改系统文件方法设定系统最大的监听队列长度)√

通过`**listen(fd, backlog)**`或系统参数（如`/proc/sys/net/core/somaxconn`）设置。
10、守护进程须关闭控制终端;√
11、服务端崩溃时,会给对等端发送RST报文; √

12、 TCP和UDP不能使用相同端口号。x

TCP和UDP端口独立，如`53`端口可同时用于TCP（DNS区域传输）和UDP（DNS查询）。

2022-2023

1、唯一标识通信的一方的三个参数是地址族、IP地址和端口号。x

唯一标识网络通信的一方
（本地协议，本地IP，本地端口）

2、使用vfork()函数生成子进程,父进程先执行,执行完后子进程再执行。x **子先**

3、UDP 在调用了connect()函数之后不能再使用sendto()函数。x

UDP调用`connect()`后可以继续使用`sendto()`，但此时`sendto()`的目标地址会被忽略（数据会发送到`connect()`指定的地址）

4、pthread_join ()函数可以等待任意线程的结束。

`pthread_join()`只能等待特定当前的线程（通过线程ID指定），不能等待分离和守护线程。

5、在多线程并发服务器中,主线程在创建了子线程之后应该关闭已连接套接字,子线程应该关闭监听套接字。√

6、调用bind)函数绑定地址时,可以指定地址和端口号,也可以只指定其中之一。√

7、调用 listen()函数的作用是 sock创建的被动套接字转换成主动套接字,主动接受来自客户端的连接请求。x **反了**

8、IO复用中在每次调用select)函数之前，都必须对描述字集合进行初始化和设置。√

9、对于数据报协议，recvfrom()返回0值，表示对端已关闭连接。x

UDP（数据报协议）中`recvfrom()`返回0是合法数据长度（空数据包），不代表对端关闭（UDP无连接状态）。

10、一个输入操作分为两个不同的阶段,即等待数据准备好和将数据从内核拷贝到进程。√

UNIX网络编程中经典的**“两步IO模型”**

11、UDP服务器端不使用 bindO函数。x
12、信号驱动IO符合异步IO的定义。x

信号驱动IO（如 `SIGIO`）是同步IO的一种，因为数据就绪后仍需进程主动拷贝数据；异步IO（如 `aio_read()`）则由内核完成全部操作后通知进程。

2022-2023-2

1、目前的主流网络通信模型是采用的OSI七层网络模型;x

实际广泛使用的是 **TCP/IP四层模型**（应用层、传输层、网络层、网络接口层），OSI七层模型仅为理论参考。

2、常量 AF_INET 协议是指的ipV4和 ipv6的协议标准;x
3、如果函数connect失败，则套接字可以继续使用。x
4、close函数用于关闭套接字符，这个函数将会立即会引发TCP的中止连接操作。√

`close` 会触发TCP四次挥手（正常终止连接），但若设置 `SO_LINGER` 可改变行为（如强制立即终止）

5、 shutdown(int sockfd, int howto)函数，可以根据参数howto关闭指定方向的数据传输。√

`howto` 可选：

- `SHUT_RD`（关闭读）
- `SHUT_WR`（关闭写）、
- `SHUT_RDWR`（双向关闭）。

6、gethostbyname（）函数可以将主机的域名转换成主机的合法IP地址。√
7、IO复用技术可以实现并发服务器。√
8、标识线程专用数据的关键字key是进程惟一的。√

9、wait()函数可以处理同时退出的多个进程。x
10、pthread _join()函数可以工作在阻塞模式下，也可以工作在非阻塞模式下。x

11、使用互斥锁可以保证，在同一时间内，只允许一个线程访问共享数据。√
12、设置SO_REUSEADDR套接字选项的描述符，可以重复绑定到同一个套接字地址结构。√

`SO_REUSEADDR` 允许绑定处于 `TIME_WAIT` 状态的地址

网安2021-2022

1、**gethostbyname**

该函数既可解析IPv4地址，也可解析IPv6地址；
该函数既可接收域名，也可接收点分十进制参数 √
当hostname为点分十进制时，函数并不执行网络查询，而是直接将其拷贝到结果字段中。

2、tcp客户端若connect（）调用失败，套接字不能再使用，必须关闭连接 √

如果 TCP 客户端的 `connect()` 调用失败（返回 `-1`），通常需要关闭该套接字并重新创建新的套接字才能再次尝试连接。

3、在多进程程序中，父进程不能先与子进程退出

父进程 **可以** 先于子进程退出，但如果不正确处理，可能会导致 **僵尸进程（Zombie）** 或 **孤儿进程（Orphan）** 问题。

4、互斥锁

5、sin_port `sin_port`（`struct sockaddr_in` 中的端口号）必须以 **网络字节序（大端序）** 存储

6、TSD 在进程中分配一个关键字（键），关键字是进程内部唯一的，用来标识一个线程专用数据（每个线程专用数据和这个关键字关联）

2021-2022

3、ICMP是信息通信管理协议的缩写;x
5、listen函数中的backlog 参数指明了两个等待队列的相关长度;√

`backlog` 定义 **已完成连接队列（ESTABLISHED）** 和 **未完成连接队列（SYN_RCVD）**

6、bind函数可使用地址复用;√
7、IO服务模型中阻塞方式是一种异步方式:x

8、selcct函数可作为定时器使用;√

`select` 可通过 `timeout` 参数实现超时等待

9、不同的协议可以绑定相同的端口;√

TCP和UDP端口独立（如DNS同时用TCP/53和UDP/53）。但同协议（如两个TCP服务）不能绑定相同IP+端口。

10、进程是比线程更小的执行单位;x
11、主线程创建子线程时应指明子线程的函数入口:√

`pthread_create` 需传入线程函数指针（如 `void* thread_func(void*)`）。

12、gethostbyname函数主要功能是通过主机名获取主机的iP相关地址信息。√

该函数解析域名返回 `hostent` 结构（含IP列表），但已过时，推荐用 `getaddrinfo`。

程序背模板

生产者-消费者问题

```
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];  // 缓冲区
int count = 0;            // 当前缓冲区中的数据数量

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_producer = PTHREAD_COND_INITIALIZER;  // 生产者条件变量
pthread_cond_t cond_consumer = PTHREAD_COND_INITIALIZER;  // 消费者条件变量
void* producer(void* arg) {  // 生产者线程函数
    int id = *(int*)arg;
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&mutex);
        while (count == BUFFER_SIZE) {// 如果缓冲区已满，等待
            printf("生产者 %d: 缓冲区已满，等待...\n", id);
            pthread_cond_wait(&cond_producer, &mutex);
        }
        // 生产数据并放入缓冲区
        buffer[count++] = i;
        printf("生产者 %d: 生产数据 %d，缓冲区大小: %d\n", id, i, count);
        pthread_cond_signal(&cond_consumer); // 通知消费者
        pthread_mutex_unlock(&mutex);
        sleep(1);  // 模拟生产耗时
    }
```

```
    return NULL;
}

void* consumer(void* arg) {
    int id = *(int*)arg;
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&mutex);
    while (count == 0) {
            printf("消费者 %d: 缓冲区为空，等待...\n", id);
            pthread_cond_wait(&cond_consumer, &mutex);
        }
        // 消费数据并从缓冲区移除
        int data = buffer[count - 1];
        count--;
        printf("消费者 %d: 消费数据 %d，缓冲区大小: %d\n", id, data, count);
        pthread_cond_signal(&cond_producer); // 通知生产者
        pthread_mutex_unlock(&mutex);
        sleep(1);  // 模拟消费耗时
    }
    return NULL;
}

int main() {
    pthread_t prod_thread, cons_thread;
    int prod_id = 1, cons_id = 1;
    pthread_create(&prod_thread, NULL, producer, &prod_id);
    pthread_create(&cons_thread, NULL, consumer, &cons_id);

    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);

    // 销毁互斥锁和条件变量
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_producer);
    pthread_cond_destroy(&cond_consumer);

    printf("主线程结束\n");
    return 0;
}
```

函数精简版

概率高:

FD_ZERO:清空fd_set描述符集合

FD_SET:往fd_set中添加描述符，建立文件描述符与fd_set联系

FD_CLR:在fd_set中删除描述符

FD_ISST:检查fd_set联系的文件描述符fd是否可读写

简答题tcp/udp流程应该**必考**

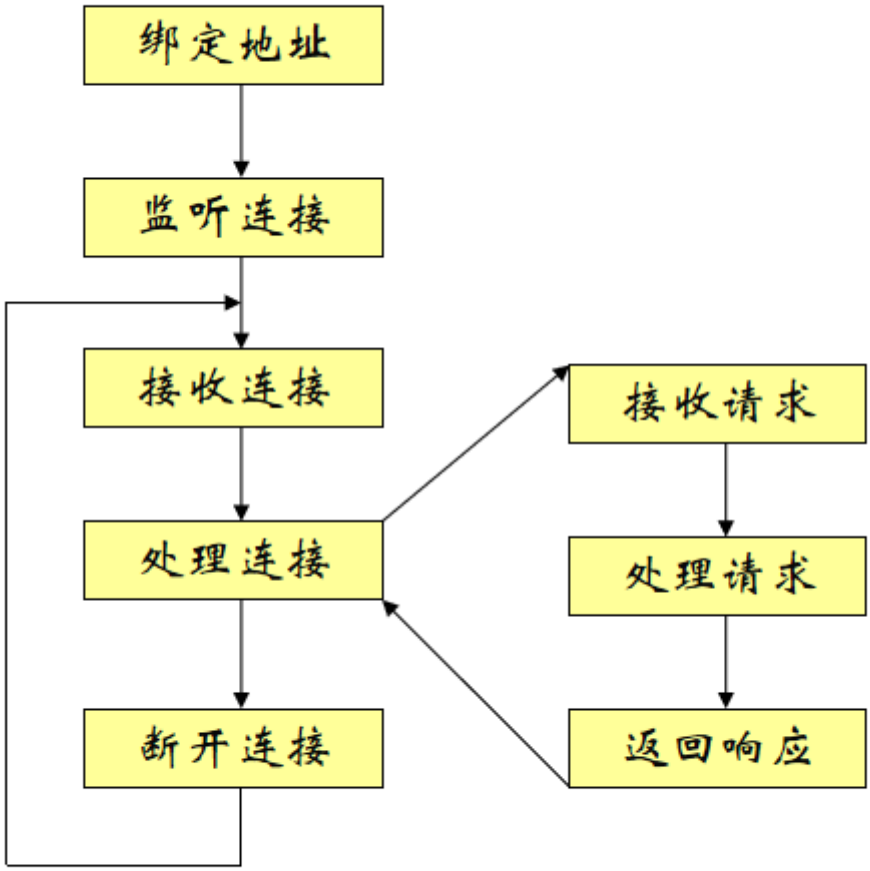
TCP套接字

TCP套接字实现过程

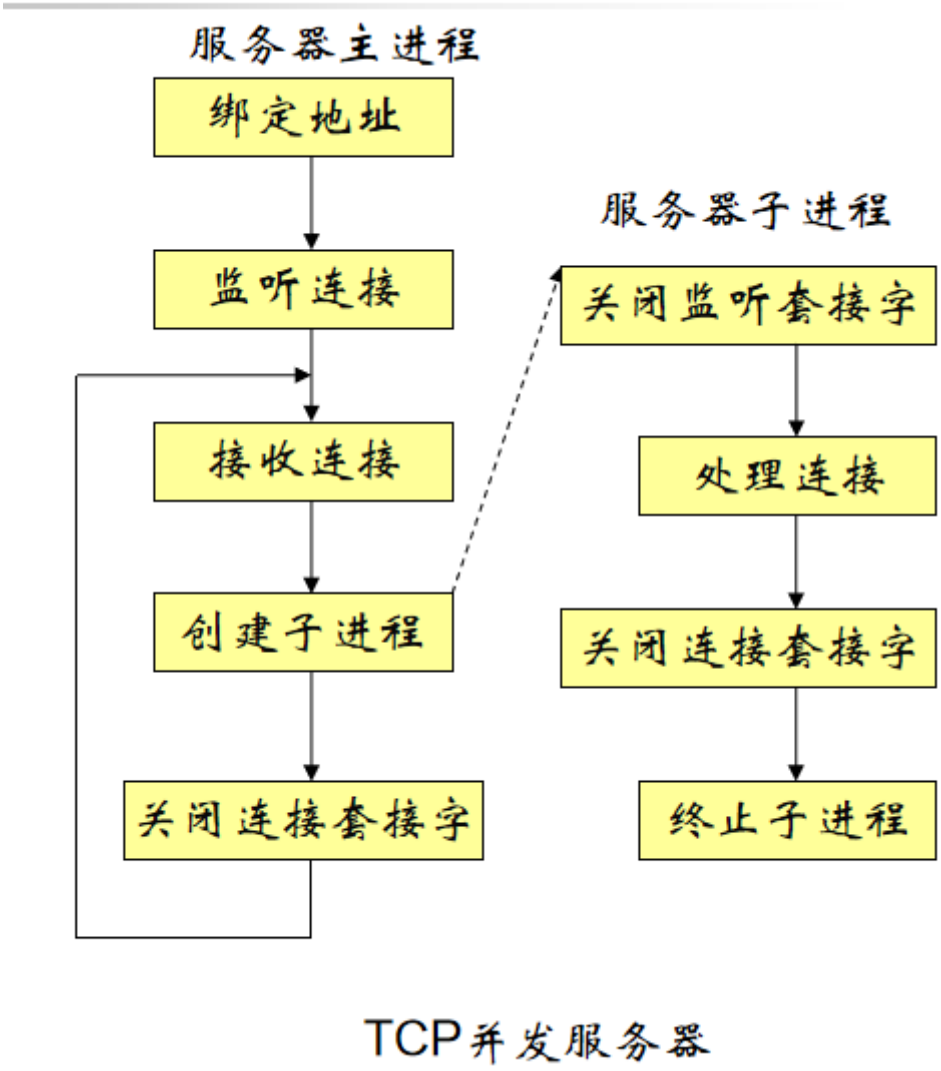
- **服务器端步骤**
  创建套接字
  绑定套接字
  设置套接字为监听模式，进入被动接受连接请求状态
  接受请求，建立连接
  读/写数据
  终止连接
- **客户端步骤**
  创建套接字
  与远程服务程序连接
  读/写数据
  终止连接

UDP套接字

- **服务器端**
  建立UDP套接字；
  绑定套接字到特定地址；
  等待并接收客户端信息；
  处理客户端请求；
  发送信息回客户端；
  关闭套接字；
- **客户端步骤**
  建立UDP套接字；
  发送信息给服务器；
  接收来自服务器的信息；
  关闭套接字



TCP迭代服务器



TCP并发服务器

多进程并发服务器建立过程：
建立连接->服务器调用fork()产生新的子进程->父进程关闭连接套接字，子进程关闭监听套接字->子进程处理客户请求，父进程等待另一个客户连接。

并发服务器

3、给新线程传递参数

传递参数的普通方法

- 单个参数

```
只传递单个参数的情况：线程创建函数
void *thread_function(void *arg){
    int fd = *((int*)arg);
    printf("%d\n",fd);
}
int main(){
     int  fd = accept(……);
     pthread_t tid;
     pthread_create( &tid, NULL, thread_function,(void*)&fd) );
     pthread_join ( mythread, NULL );
}
```

- 多个参数

传递多个参数：由于线程创建函数只允许传递一个参数，因此当需要传递多个数据时，应首先将这些数据封装在一个结构体中。

```
void *start_routine(void *arg);
struct ARG {
    int connfd;
    char msg[256]; // 例如 欢迎消息
    ……            //其他需要传递的参数
};
```

通过分配arg的空间来传递参数（熟练掌握）

主线程首先为每个新线程分配存储arg的空间，再将arg传递给新线程使用，新线程使用完后要释放该空间。

arg = (struct ARG*)malloc(sizeof(struct ARG));

3、 **accept函数最多返回三个值**：一个既可能是**新套接字**也可能是**错误指示的整数**，一个客户进程的协议地址（由**cliaddr**所指），以及该地址的**大小**（这后两个参数是值 - 结果参数）；

4、sockaddr_in

```
struct sockaddr_in{
  unsigned short int sin_len;  /* IPv4地址长度 */
  short int sin_family; /* 地址类型 */
  unsigned short int sin_port; /* 存储端口号 */
  struct in_addr sin_addr;  /*存储IP地址 */
  unsigned char sin_zero[8];  /* 空字节 */
};
```

sin_family指代协议族，在TCP套接字编程中只能是AF_INET;

sin_port存储端口号（使用网络字节顺序），数据类型是一个16位的无符号整数类型；

sin_addr存储IP地址，IP地址使用in_addr这个数据结构：
  struct in_addr{ unsigned long s_addr; };
这里的s_addr按照网络字节顺序存储IP地址。

sin_zero是为了让sockaddr与sockaddr_in两个数据结构保持大小相同而保留的空字节。

可能考一些细碎知识

1、 TCP和UDP对比

TCP（Transmission Control Protocol）可靠的、面向连接的协议、传输效率低全双工通信（发送缓存&接收缓存）、面向字节流。使用TCP的应用：Web浏览器；文件传输程序。
UDP（User Datagram Protocol）不可靠的、无连接的服务，传输效率高（发送前时延小），一对一、一对多、多对一、多对多、面向报文(数据包)，尽最大努力服务，无拥塞控制。使用UDP的应用：域名系统(DNS)；视频流；IP语音(VoIP)。

2、tcp三次握手四次挥手老生常谈

3、 TCP连接的建立

几个函数打开和SYN分节解释
服务器必须准备好接受外来的连接。通过调用socket, bind, listen函数完成。称为被动打开。
客户通过调用connect进行主动打开。这引起客户TCP发送一个SYN分节，告诉服务器客户将在连接中发送的数据的初始序列号。

服务器必须确认客户的SYN，同时自己也得发送一个SYN分节。服务器以单个分节向客户发送SYN和对客户的SYN的ACK。
客户必须确认服务器的SYN。

4、 TCP一般用四个分节终止一个连接

某个进程首先调用close, 这一端的TCP于是发送一个FIN分节，表示数据发送完毕。主动关闭。
另一端称为被动关闭。TCP对接收的FIN分节进行确认，并以文件结束标志传递给应用程序。
一段时间后，接收到文件结束标志的应用程序调用close，这也导致向对方发送一个FIN分节。
接收到这个FIN分节的原发送方TCP对它进行确认。

5、调用wait或waitpid函数时，正常情况下，可能会有以下几种情况：
阻塞（如果其所有子进程都还在运行）；
获得子进程的终止状态并立即返回（如果一个子进程已终止，正等待父进程存取其终止状态）；
出错立即返回（如果它没有任何子进程）

6、僵尸进程：由于linux信号不排队，在SIGCHLD信号同时到来后，信号处理程序中调用了wait函数，其只执行一次，这样将留下2个僵尸进程

7、多进程并发服务器建立过程：
建立连接->服务器调用fork()产生新的子进程->父进程关闭连接套接字，子进程关闭监听套接字->子进程处理客户请求，父进程等待另一个客户连接。