# 简答

1. **TCP echo 时序**

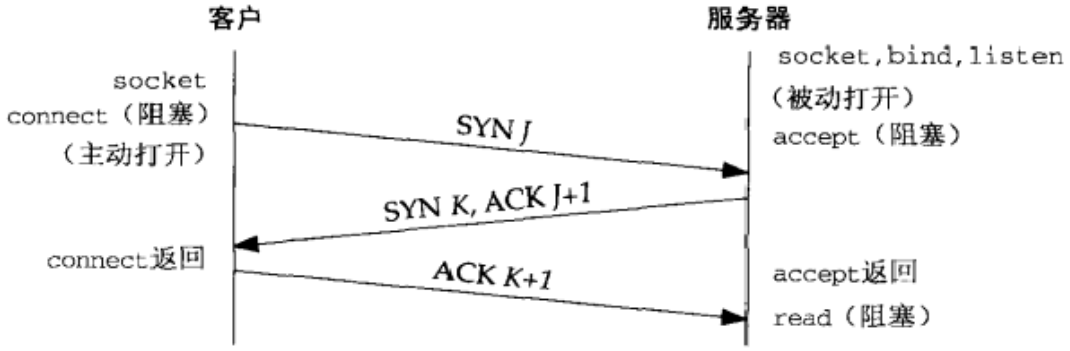


图2-2　TCP的三路握手

2. **僵尸进程，同卷一**
3. **TCP实现并发服务器**

   创建listenfd，绑定端口，用listen函数将套接字转化为监听套接字，然后使用accept函数阻塞进程，当客户端连接请求到来时，用connfd接收accept函数的返回值，然后创建子进程，用pid判断当前是父进程还是子进程，如果是父进程就关闭connfd，如果是子进程就关闭listenfd然后由子进程对该客户服务

4. **TCP三次握手图例包括系统调用**

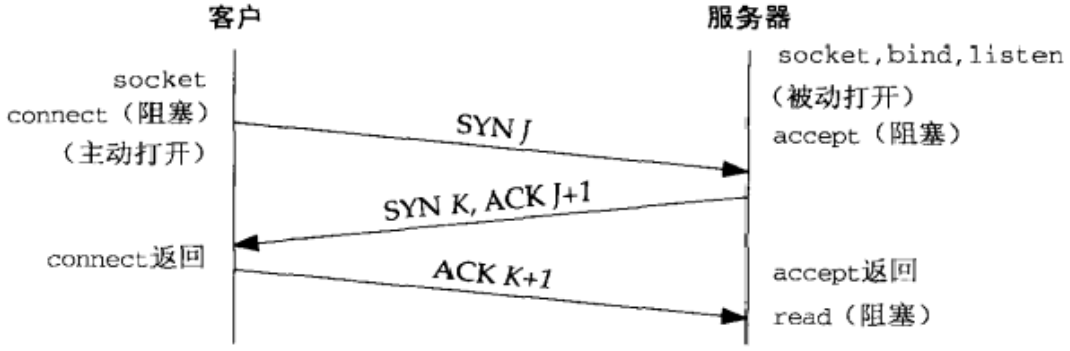


图2-2　TCP的三路握手

5. **解释traceroute工作原理**
   - traceroute 可以用来测量本机到一个目的主机的路径
   - 实现的原理是利用了IPv4的TTL字段或IPv6的跳限字段，一开始设置为1，然后中间节点会返回一个ICMP”time exceeded in transmit”（传输中超时）错误，接着逐渐增大TTL，从而逐步确定下一跳路由地址。直至目的节点返回一个ICMP”port unreachable”（端口不可达）错误，则表示到达目的节点（这要求目的节点没有在该端口上开启服务，即发送ICMP包时，目的端口号应该选择一个未被目的主机使用的端口号 ==> traceroute选择了一个大于30000值作为目的端口号，因为UDP协议要求端口号必须小于30000，所以目的主机如果接收到必然会会一个ICMP端口不可达错误）。
6. **五种IO模型，select属于I/O复用模型**

## 阻塞式I/O模型

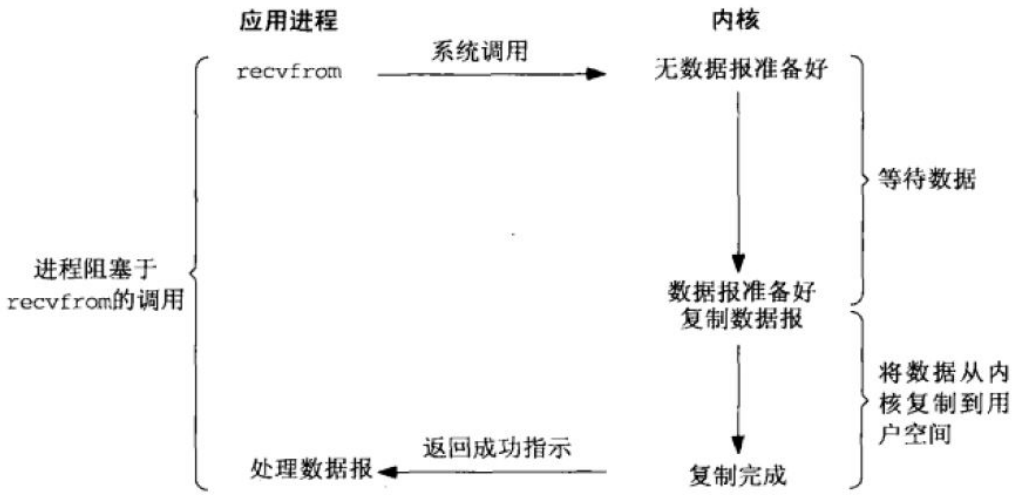


图6-1　阻塞式I/O模型

阻塞式I/O模型

## 非阻塞式I/O模型 ==> 轮询（polling)

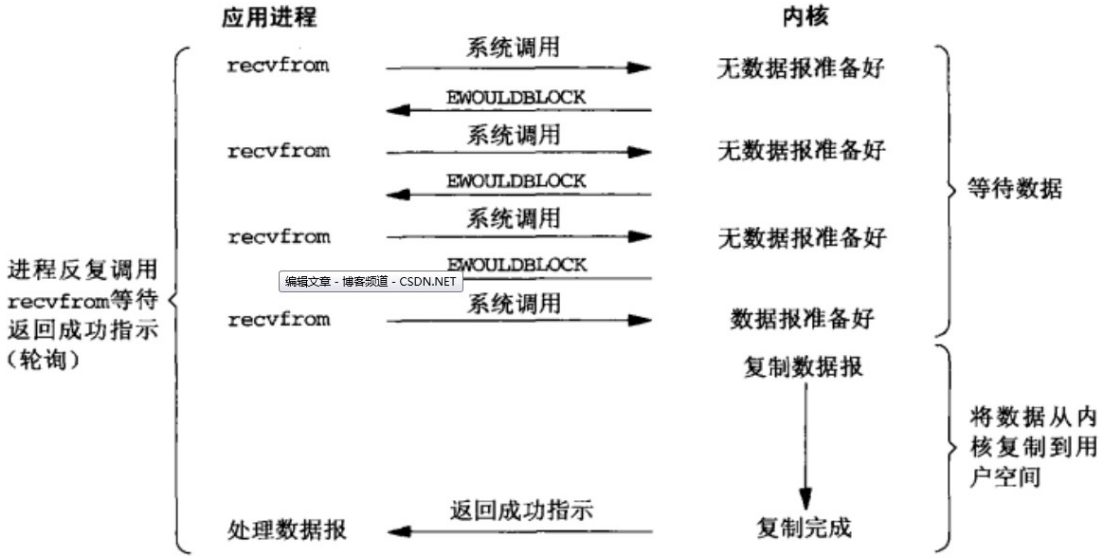


图6-2　非阻塞式I/O模型

非阻塞式I/O模型

## I/O复用模型

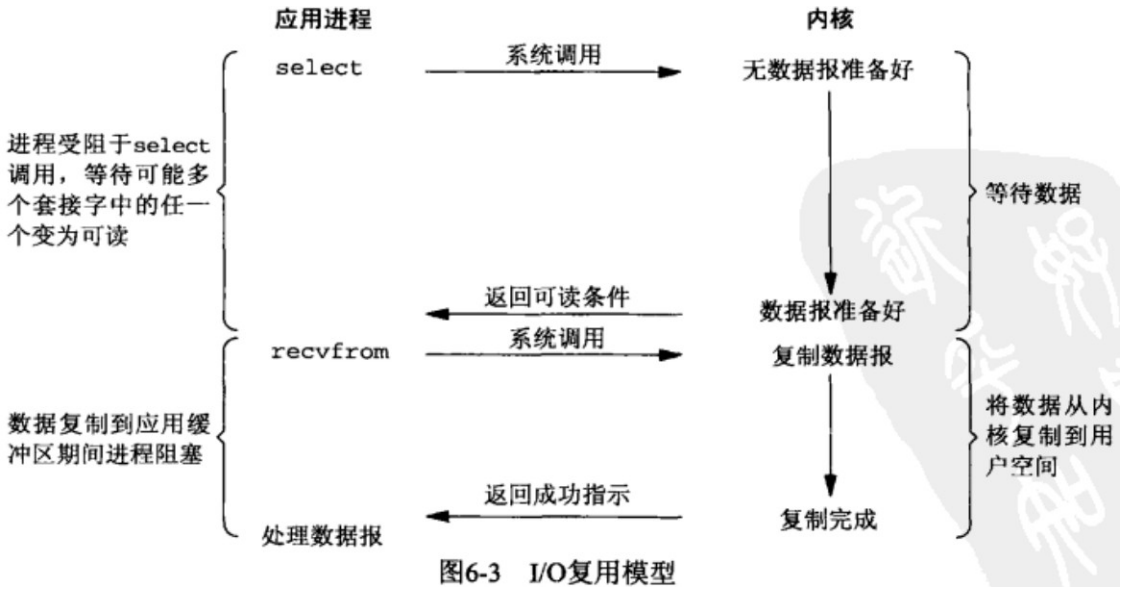


图6-3 I/O复用模型

I/O复用模型

## 信号驱动式I/O模型

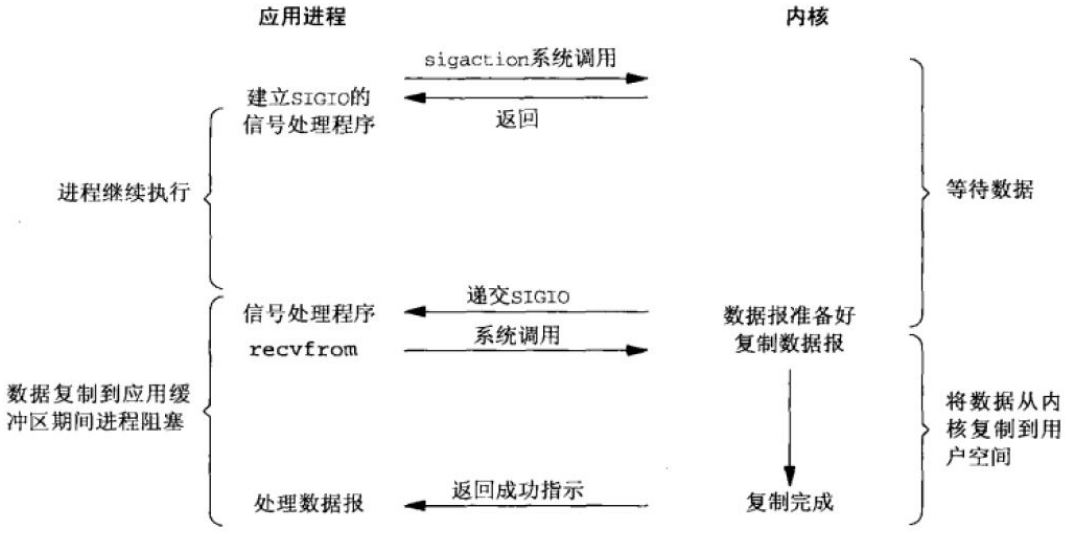


图6-4 信号驱动式I/O模型

信号驱动式I/O模型

## 异步I/O模型

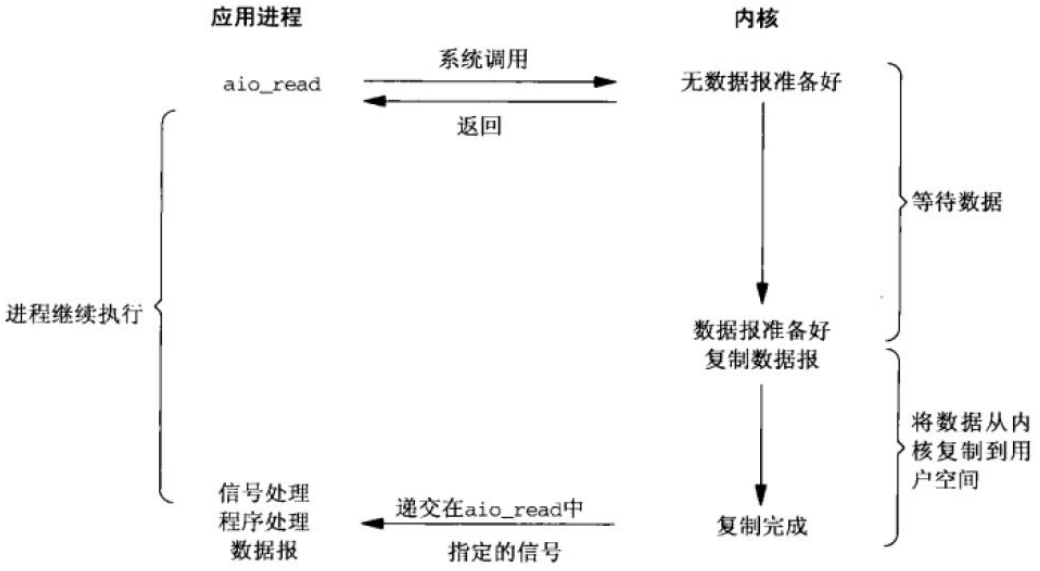


图6-5　异步I/O模型

异步I/O模型

## 各种I/O模型的比较

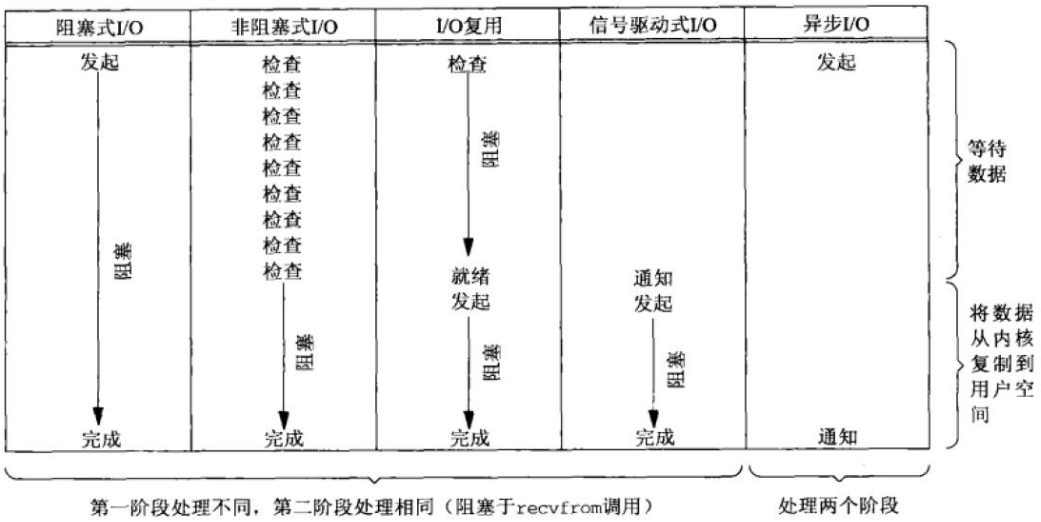


图6-6　5种I/O模型的比较

5种I/O模型的比较

- 同步I/O操作（synchronous I/O operation）：导致请求进程阻塞，知道I/O操作完成
- 异步I/O操作（asynchronous I/O operation）：不导致请求进程阻塞
- 上述5种模型中，前4种是同步的，最后一种是异步的

# 代码分析

1. **解释getservbyname（）的三种情况**

```
sptr = getservbyname(”domain“, “udp”); //成功，DNS服务信息
 sptr = getservbyname(”ftp“, “NULL”);  //成功
  sptr = getservbyname(”ftp“, “udp”);  //出错，返回NULL
  通过服务名获取服务信息（“服务名”，“协议名”）
```

```
sptr = getservbyname("ftp", "tcp");//FTP using TCP
sptr = getservbyname("ftp", NULL); //FTP using TCP
这两个结果是一样的
```

2. **TCP echo程序如何区分服务器进程终止和服务器崩溃**

   **进程终止：**客户端收到一个FIN并返回一个ACK，但客户进程此时处于fgets调用的阻塞中，等待从客户端接收到一行文本。（此时客户端并不知道服务器子进程已经终止，只是单纯的接收到了一个FIN）而后，我们在客户上再键入一行文本。str_cli调用writen发送数据给服务子进程，但由于服务子进程已经关闭，于是发送一个RST。我们客户端readline时若先收到第二步中的FIN，则立即返回0，如果先接收到RST，则返回一个ECONNRESET（对方复位连接错误）

   **服务器崩溃：**在客户端上键入一行文本，由writen写入内核。 writen成功返回后，客户随后阻塞于readline调用，客户端持续重传数据分节，试图从服务器上接收一个ACK。然而服务器已崩坏，没有任何响应，最终readline调用上返回一个错误。如果是主机崩溃，则返回超时ETIMEOUT

   进程终止：服务器发送FIN

   崩溃： 服务器不会在已有网络上发送任何信号

# 编程

1. **wait**
2. 写出重启被中断的accept函数程序片

   教材p108

```
for(;;)
{
    clilen= sizeof(cliaddr);
    if(connfd=accept(listenfd,(struct sockaddr *)&cliaddr, &clilen) <0 )
    {
        if(errno == EINTR)
            continue;
        else
            err_sys("accept error");
    }
}
```

3. 解释datalink access，写出一个datalink access的实现

   操作系统为应用程序提供访问数据链路层的功能，这种功能可提供如下能力：

   - **能够监视由数据链路层接收的分组**
   - 能够**作为普通应用进程**而不是内核**的一部分运行某些程序**，例如：RARP

```
fd=socket(PF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
```

4. 改IP包TTL用什么函数，用哪个socket option

```
setsockopt(int sockfd, int ! ! , int TTL, const &ttl,socklen_t optlen )
```

```
#include <sys/socket.h>
```

```
/**
* 获取一个打开的套接字的选项
*
* @param sockfd       必须指向一个打开的套接字描述符
* @param level        级别
* @param optname      选项名
* @param optval       指向一个变量的指针，用于接收函数的结果，其长度由最后一个长度限定
* @param optlen       这是一个Value-Result参数，传入时限定optval的最大长度，防止缓存
溢出
*                     函数执行结束时，可以通过这个参数知道内核究竟往optval写了多少数据
* @return 成返回0，出错返回-1
*/
int getsockopt(int sockfd, int level, itn optname, void *optval,
               socklen_t *optlen);

/**
* 设置一个打开的套接字的选项
*
* @param sockfd       必须指向一个打开的套接字描述符
* @param level        级别
* @param optname      选项名
* @param optval       指向一个变量的指针，用于向函数传递要设置的值，其长度由最后一个长度
限定
* @param optlen       指示了optval的长度
* @return 成返回0，出错返回-1
*/
int setsockopt(int sockfd, int level, int optname, const void *optval,
               socklen_t optlen);
```