**lec1 intro**

Q1：什么是面向连接的？什么是无连接的？各自有什么特点？

面向连接的：双方在传输数据之前必须（通过某种协议）建立起一个逻辑或物理的数据通道。通常是可靠的、稳定的、但占用较多的资源

无连接的：每个数据报都附带了足够的目的地信息，进行独立传输。通常是不可靠的，但性能较高。

Q2：网页浏览是面向连接的还是无连接的？为什么？收发Email呢？

网页浏览使用的是HTTP/HTTPS协议，是面向连接的，因为它们都是基于TCP协议。TCP协议在传输数据之前需要先建立连接，确保数据的可靠性，建立连接后，数据才能开始传输；当数据传输完成后，连接才会被断开

收发邮件使用的是SMTP、IMAP和POP3等协议，是面向连接的，因为它们也是基于TCP协议，在发送和接收邮件时，邮件服务器会与客户端建立一个TCP连接，然后通过这个连接传输邮件数据。

Q3：哪种套接字是面向连接的？哪种是无连接的？

流式套接字面向连接

数据报套接字、原始套接字无连接

Q4：C/S模型有哪些优点和缺点？（相对于对等模型）

优点：

职责明确，易于维护，服务器可在客户不知晓、不改变的情况下完成升级、替换

数据一般集中存放在服务器端，易于采取安全措施和权限控制

管理员易于对数据进行升级，对等模式可能要更新成千上万个节点

技术成熟，易于管理

缺点：

服务器端负载重，容易造成网络拥堵；对等模型可以将负载均摊到各个节点

资源集中存放，健壮性较差；对等网络中资源分布在多个节点，某几个节点出现故障不影响其它节点使用

服务器易称为黑客集中攻击的目标

Q5：为什么在可执行文件前加./？

linux系统中，执行文件的搜索路经被定义在环境变量“PATH”中，如果可执行文件不在PATH指定目录下，系统无法直接找到它。通过使用“./”告诉系统，在当前目录下查找文件，而不是在“PATH”中定义的目录中查找

将文件所在路经加入环境变量PATH中

**lec2 tcpsocket**

Q1:struct sockaddr\_in sa;

sa.sin\_addr和sa.sin\_addr.s\_addr的区别？哪个表示了最后的IP地址值？

sa.sin\_addr是一个结构体，用来存储IP地址，sa.sin\_addr.s\_addr是sa.sin\_addr中的一个字段，它是一个无符号整数，表示实际的IP地址值

Q2：TCP/IP的网络字节序是big endian还是little endian，为什么？

TCP/IP采用big endian排序方式，因为TCP/IP规定把接收到的第一个字节当作高位字节看待，这就要求发送端发送的第一个字节是高位字节。

并且发送端发送数据时，发送的第一个字节是该数值在内存中起始地址处对应的字节，因此高位字节要放在低地址处。

因此，网络字节序是big endian

Q3：listen请求队列中的连接请求是否已经完成TCP三次握手？

请求队列中的连接已经完成了三次握手，被TCP接受，但还没有被应用层所接受。TCP接受一个连接是将其放入队列，应用层接受连接是将其从队列中移除

Q4：对于新的连接请求，如果队列中已经没有任何空间，TCP将不会理会收到的SYN，也不会发送任何报文段，为什么不回送RST呢？

这是一个软错误，通常队列满是由于应用程序或操作系统忙造成的，这个条件在一个很短的时间内就可以改变，如果回送RST会将客户进程的主动打开废弃，不如让它重发SYN效率高。

Q5：accept新创建的socket的端口号和原sockfd的端口号是否相同？

相同，主监听套接字和新创建的套接字状态不同（listen/established），处于established的套接字不能接受SYN报文段，而处于listen的套接字不能接收数据报文段。四元组区分TCP套接字连接：通常处于listen状态的主套接字的daddr和dport都是0。

Linux中处于listen状态的socket和处于established状态的socket分别位于两个hash table中，而且是后搜索listen状态的hash table

Q6：shutdown howto三个值都会如何？

howto=0：关闭读通道，丢弃尚未读取的数据。对接收到的数据发送ack后丢弃

howto=1：关闭写通道，继续发送发送缓冲区中未发送完的数据，然后发送FIN字段关闭写通道

howto=2：关闭写通道，任何进程不能再操作这个socket

Q7：shutdown和close的区别

1.关闭对象和影响范围不同：shutdown关闭连接通道，所有进程不能再使用已被关闭的通道；close关闭本进程的sockfd，但链接仍然开着，引用这个sockfd的其它进程仍然可以使用这个链接

2.关闭粒度不同：close关闭进程与socket的整个接口，调用close之后本地进程不能再读写这个socket；shutdown可以只关闭一个方向的通道，另一个方向的通道仍然可以操作

Q8：write和close相关

write函数成功返回并不意味着数据已经通过网络发送到对方主机，而只是说明数据从进程缓冲区发送，并已存放在套接字发送缓冲区中，等待TCP协议来发送

close后本进程不能再使用这个套接字，但TCP可能并没有删除套接字结构，因为可能有其它进程还在使用它。调用close只是将对sockfd的引用减1，直到对sockfd的引用为0时才清除sockfd

Q9：TCP在小数据包是否延迟发送中，有TCP\_NODELAY选项，其中使用了Nagle算法，什么是Nagle算法？

TCP协议使用的，在目的端发送多个小数据包时，延迟发送，把小数据包装在一起，组成大数据包再发送的算法

Q10：fcntl(fd, F\_SETFL, O\_NONBLOCK)这样设置socket为非阻塞对吗？为什么？如果不对应该如何设置？

不正确。因为flags的每一位都有对应的意义，如果直接使用O\_NONBLOCK会使socket配置的其它设置失效，因此只需要更改O\_NONBLOCK对应位置的参数值即可。

int flags;

flags = fcntl(fd, F\_GETFL, 0);

flags **|=** O\_NONBLOCK;

fcntl(fd, F\_SETFL, flags);

ps:如果要变为阻塞

int flags;

flags = fcntl(fd, F\_GETFL, 0);

flags **&= ~**O\_NONBLOCK;

fcntl(fd, F\_SETFL, flags)

Q11：这样设置socket为非阻塞对吗？

Int nIO=1;

ioctl(fd, FIONBIO, &nIO);

对

Q12：DNS使用什么协议向前转发？

SMTP

Q13：DNS域名查找通常使用UDP协议，服务器端口号是多少？

53

lec3 udp & raw socket

Q1：UDP协议不保证数据报按序到达，如果应用程序需要按序，该怎么做？

发送方对每个数据包进行编号，接收方收到数据报后，根据序号对其进行处理，不符合顺序则必须等待

Q2：UDP协议没有流量控制，可能造成什么结果？怎么避免？

如果发送速度大于接收速度，当套接字接收缓冲区满后，之后到达的数据报将被丢弃。

简单流控机制：由程序维护一个发送缓冲区，将数据报保存在该缓冲区，直到收到确认才清除，当用户缓冲区满时不再发送数据报

Q3：原始套接字相对于TCP和UDP套接字的优缺点是什么？

优：对通信协议的支持范围广，能够访问ICMP、IGMP等多种协议的数据报，加强了用户对数据的操作能力，对IP数据报操作更加灵活，允许读写首部在哪的IP数据包

缺：编程接口不够简化，编程操作相对复杂，只能由超级用户或系统管理员创建

Q4：四种TCP体制

lec4 Linux进程与信号机制

Q1：进程和程序有什么区别？

进程是动态的，程序是静态的，进程由程序产生。

程序是一个静态的命令集合，不占系统的运行资源；进程是一个随时可能变化的、动态的、使用系统运行资源的程序，而且一个程序可以启动多个进程

Q2：Linux进程的虚拟地址空间是多少？

32位下位0～4G-1，其中最高的1G字节被称为“内核空间”，由所有进程共享，进程通过系统调用进入内核；较低的3G由进程使用，称为“用户空间”。每个进程都有4G的虚拟地址空间

Q3：Linux进程分类

交互进程：Shell启动和控制的进程

批处理进程：不属于某个终端，被提交到某个序列中，按顺序执行

守护进程：后台运行的进程，需要时被唤起，通常在系统启动时开始执行

专用进程：进程号为0的调度进程，在内核态运行，又称为系统进程，整个系统活动期间生存，不会被终止

Q4：理解fork

两次返回：子进程中，fork返回0，表示是子进程；父进程中返回正整数，表示子进程id

创建fork时：系统创建新进程，为该进程准备数据段、堆栈段和代码段。数据段和堆栈段从父进程复制；和父进程使用相同的代码段

Q5：fork后，为什么子进程只会执行fork后的代码？

因为子进程继承了父进程的指令指针IP/EIP/RIP

Q6：写时拷贝

fork结束后不会立刻复制父进程的内容，而是到真正使用的时候再复制，这样如果下一条语句时exec（），fork就不会白复制了

Q7：典型守护进程有什么特点？

通常在系统初始化时被启动、生存期为系统执行时间、一直等待事件发生并处理事件、可以利用其它进程完成各种请求、一般不和终端联系

Q8：设计xinetd守护进程的原因

几乎每个服务进程的初始化过程均相同

每个服务进程在系统进程表中占据一项，但大部分时间，这些服务进程处于睡眠状态

Xinetd为大量服务创建套接字，并利用select监听所有服务，在远程主机请求某项服务时，为该端口生成特定服务器

（dup2函数重定向输入输出）

Q9：解释一下 void (\*signal(int signum, void (\*handler)(int))) (int)

函数signal有两个参数，一个是int signum，执行需要捕获的信号；一个是函数指针handler，指定信号处理函数，该函数有一个int参数，返回值为void类型；函数signal返回一个函数指针，这个函数指针指向的函数有一个int参数，返回值为void类型

lec5 进程间通信

Q1：主要进程间通信方式

管道和命名管道：管道可用于具有亲缘关系进程间的通信，命名管道允许无亲缘关系进程间的通信

消息队列：消息的链接表

共享内存：使得多个进程可以访问同一块内存空间，速度最快，常与信号量结合实现进程间的同步及互斥

信号量：作为进程间及同一进程不同线程之间的同步手段

套接字：因特网套接字主要面向不同主机进程间通信，UNIX域套接字面向同一主机上进程间通信

Q2：消息队列、共享内存和信号灯通常被称为System V进程间通信机制，什么是System V？

是UNIX的另一个分支

Q3：信号和信号量的区别

信号是一种处理异步事件的方法，它是由硬件或软件触发，再由操作系统内核发送给应用程序的中断形式。

信号量是一种实现进程间同步、互斥的机制，是一种数据操作锁的概念

Q4：管道和消息队列需要几次数据拷贝？共享内存呢？

管道和消息队列：4次拷贝（输入 -> 发送方缓冲区 -> 内核缓冲区 -> 接收方缓冲区 -> 输出）

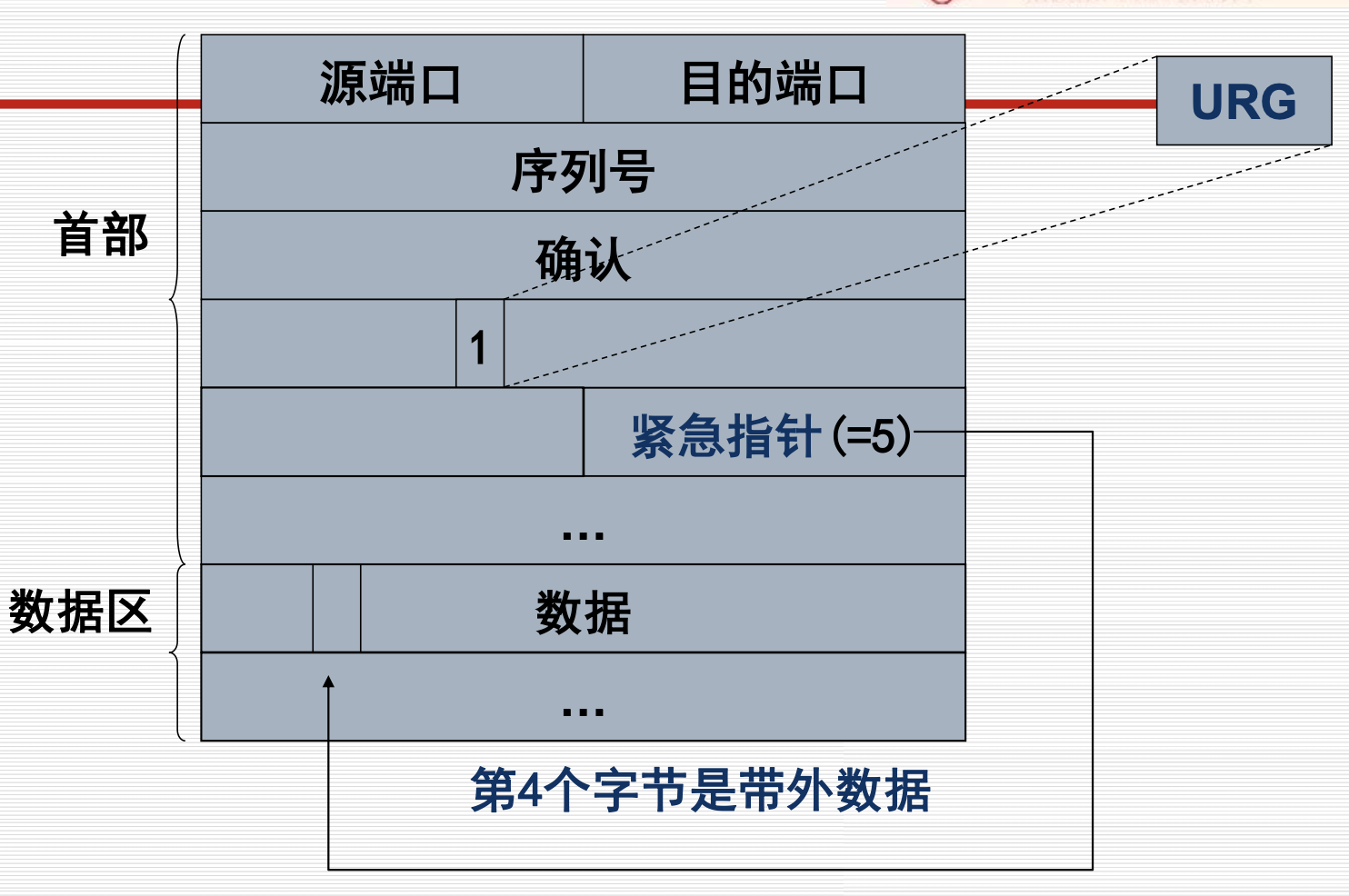
共享内存：2次拷贝（输入 -> 共享内存区 -> 输出）

Q5：UNIX域套接字与Internet套接字相比，进程间通信哪个效率更高？为什么？

UNIX域套接字更高。因为UNIX域协议由于知道数据不会离开主机，所以只需要较少的处理过程；而Internet套接字需要使用TCP/IP协议，因此数据会经过TCP/IP协议栈处理，包括拥塞控制、流量控制等，会增加额外开销

lec6 OOB

Q1：

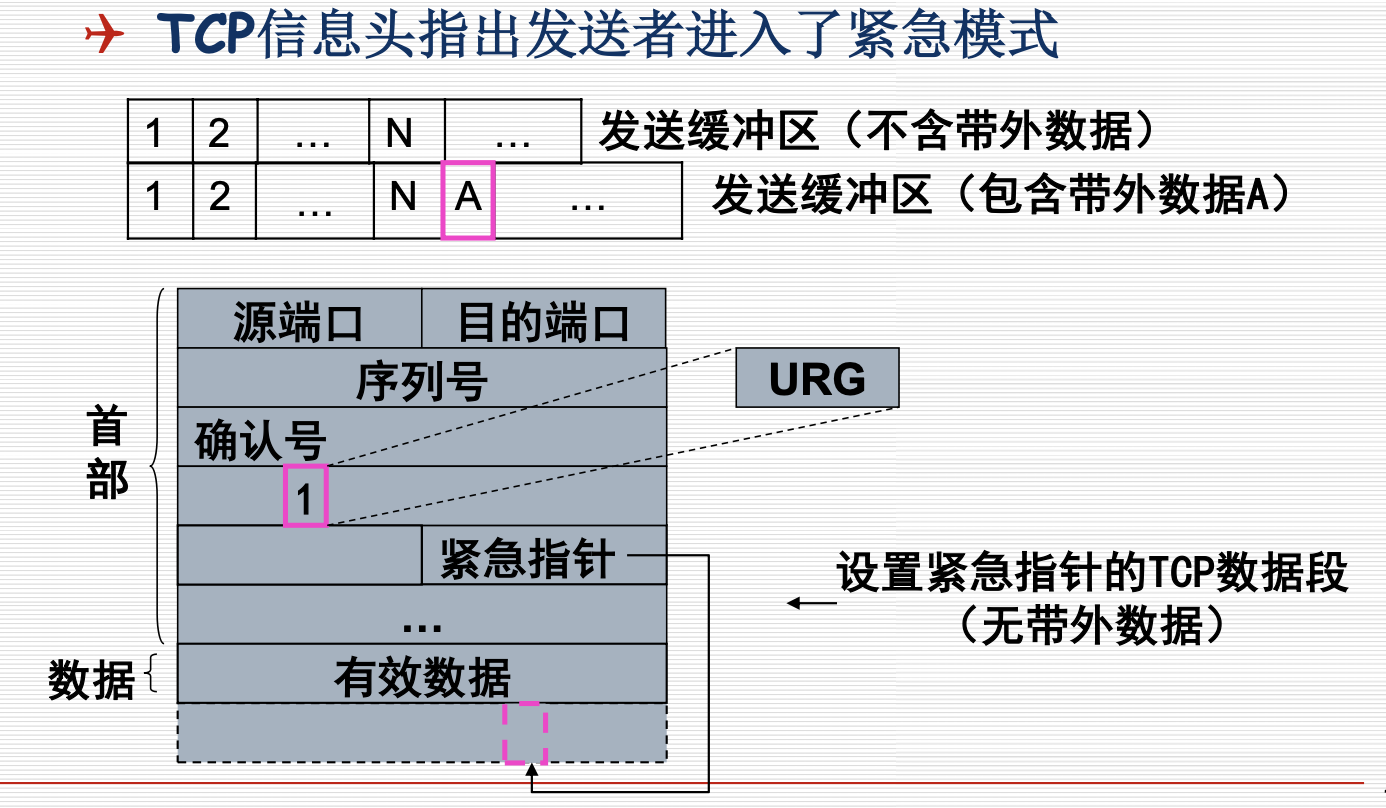


URG=1；紧急指针指向带外字节的下一个字节

Q2：TCP带外数据发送的特殊情况

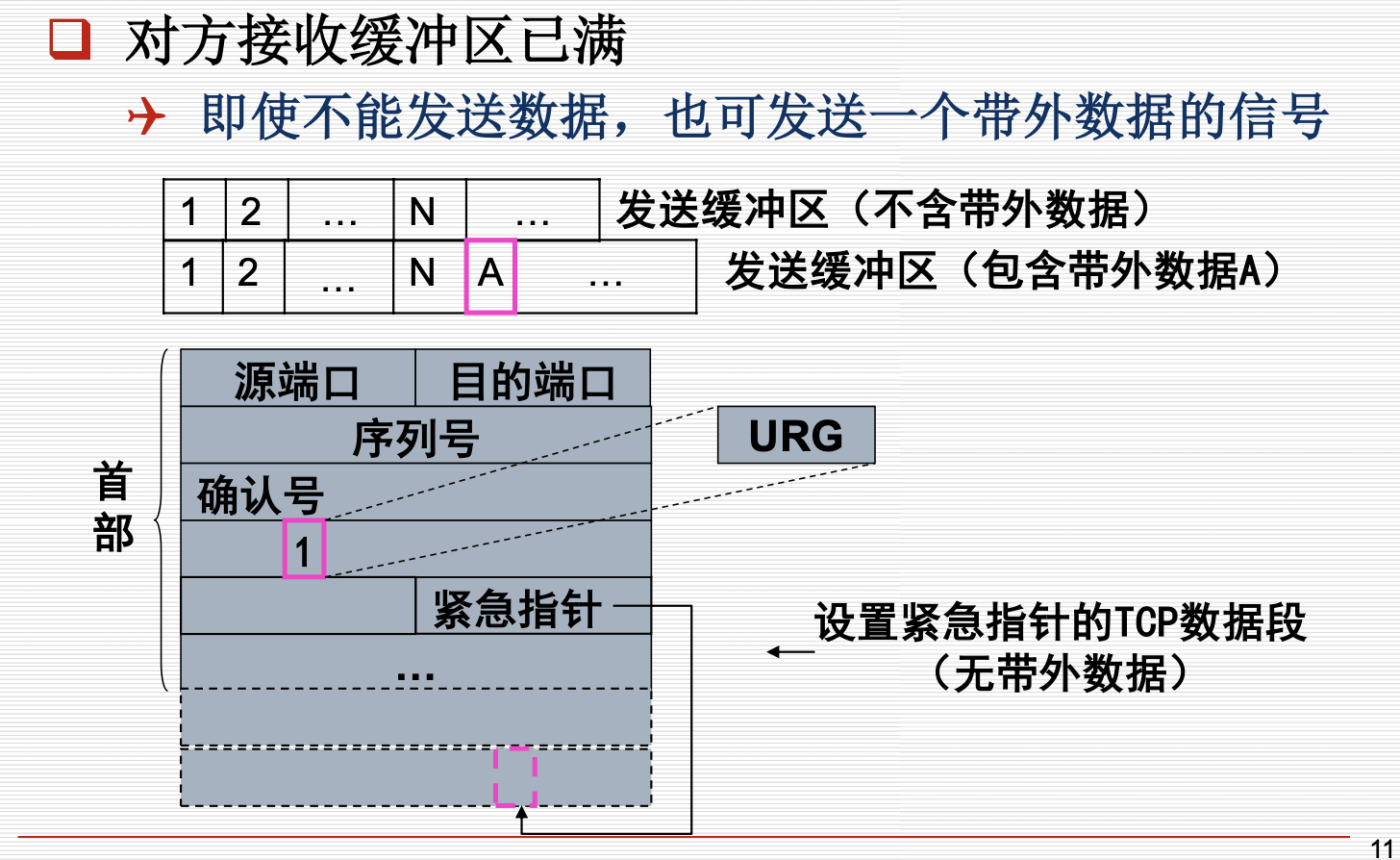
1.TCP数据段大小不够，当前数据段中无法包含带外数据

此时发送的数据段中URG=1，但不包含带外数据



2.对方接收缓冲区已满

此时发送URG=1的空数据段，并设置紧急指针为带外数据在这个发送缓冲区的位置偏移；TCP再随后发送的数据段中均设置URG标志，直到带外数据被送出



Q3：带外数据接收情况？

真正到达接收方时可能已经收到多个包含紧急指针的TCP数据段

TCP协议接收到URG标志的数据段时，TCP会检查紧急指针来判断它所指向的数据是否已经到达本地；并且会比较数据段中的紧急指针和最后一次接收的紧急指针，以确定是否指向相同带外数据

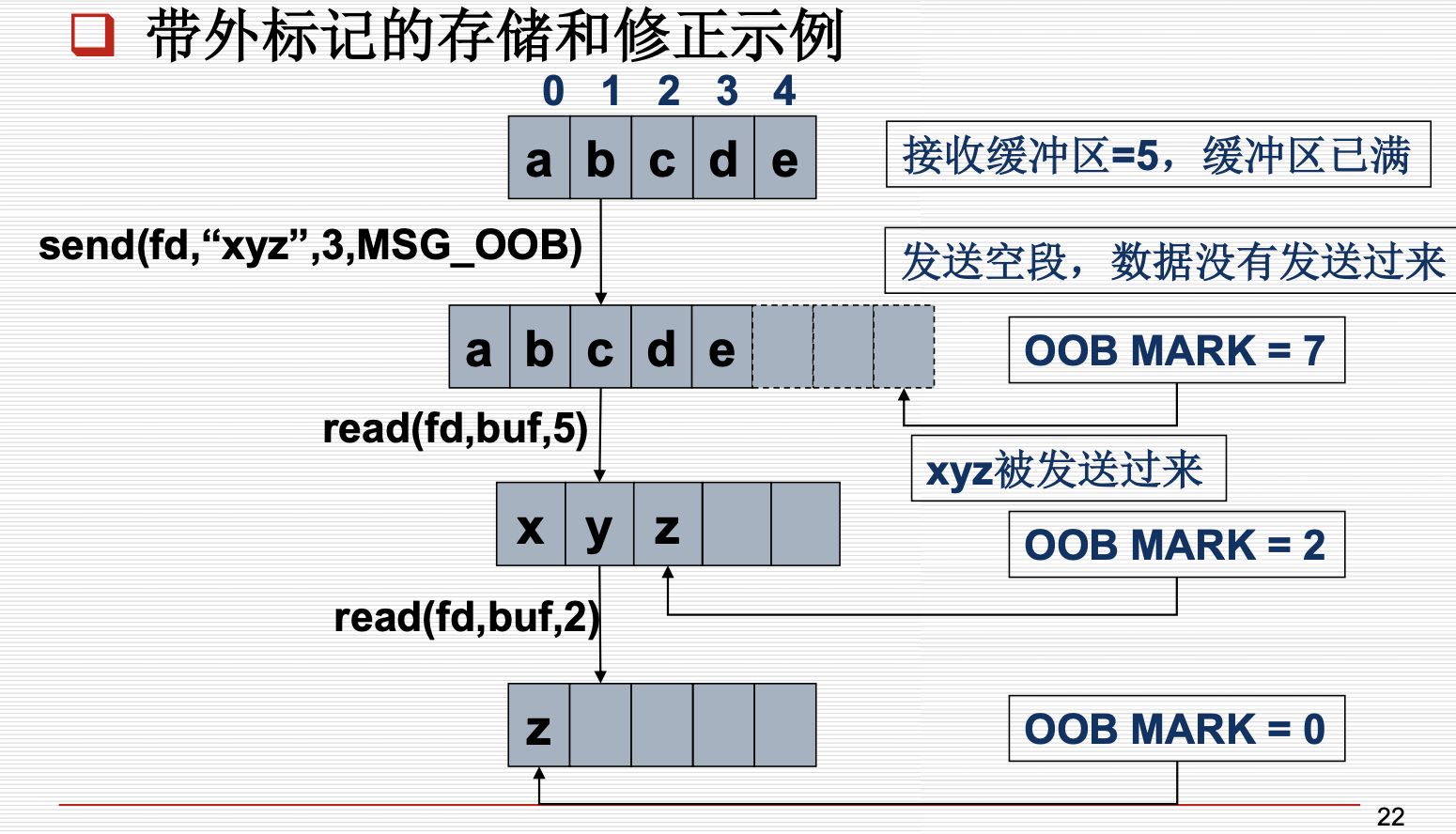
TCP可能会将数据分成多个小的数据包传输，可能有几个数据包中含有紧急指针，但他们都指向同一个位置，对于这种带外数据，虽然有多个指针指向它，但只有第一个紧急指针才会通知程序

Q4：SO\_OOBINLINE设置与否对接收的影响

未设置时，带外数据被放入单独的带外数据缓冲区中，想要接收可以使用设置了MSG\_OOB的recv、recvfrom、recvmsg等

设置时，带外数据被放入正常套接字缓冲区中，使用普通接收函数读取

Q5：带外标记



正常读取的数据将在带外标记前停止。

假设oob mark=20，缓冲区数据长度为50，调用read（fd，buf，40），返回值为20

lec7 IO

Q1：为什么UDP套接字写操作时不会阻塞？

UDP协议没有专门为UDP套接字设置发送缓冲区，当使用sendto发送数据时，该函数将要发送的数据从用户缓冲区拷贝到系统缓冲区然后返回。只要数据长度不超过最大长度，都可以成功发送

Q2：阻塞式IO模型下，tcp和udp都可能产生阻塞的过程是？

读

Q3：非阻塞IO模式下，返回错误不同的是哪个步骤？

建立连接，启动3次握手立刻返回EINPROGRESS

其它（读写接收）都为EWOULDBLOCK

Q4：多路复用IO适用场合

1.当一个客户端需要同时处理多个文件描述符的输入输出操作的时候

2.当程序需要同时进行多个套接字操作的时候

3.一个TCP服务器需要同时处理正在侦听网络连接的套接字和已经连接好的套接字

4.一个服务器同时使用TCP和UDP协议

5.一个服务器同时使用多种服务，且每种服务使用不同的协议

Q5：使用信号驱动IO的三个步骤

1.设置SIGIO的信号处理函数

Signal(SIGIO, sigio\_handler)

Siganciton(SIGIO, &act, NULL)

2.设置socket描述符的所有者

Fcntl(sockfd, F\_SETOWN, getpid())

3.允许这个socket进行信号驱动IO

Int on = 1;

Ioctl(sockfd, FIOASYNC, &on)

Q6：为什么信号驱动IO适用于UDP而TCP一般不使用？

TCP协议在很多环节都会产生SIGIO信号，难以区分产生SIGIO信号的原因；而UDP只在收到数据包或错误时产生SIGIO信号

lec8 Winsock

Q1：winsock1.1，主要扩充是实现对网络事件基于消息的异步存取

启动和终止函数；异步服务函数；异步选择函数；阻塞处理函数；错误处理函数

Q2：winsock2，为了适应32位网络编程而制定

在Winsock1.1的基础上，提供了一个与协议无关的、具有实时网络传输能力的socket

数据传输函数、名字注册解析函数

Q3：MFC将Winsock API封装，提供两类cpp对象。

CAsyncSocket：低层次封装

CSocket：高层次抽象

Q4：Winsock和socket编程的异同

相同：

Winsock从Berkeley Socket扩展而来，具有继承性

标准Winsock API函数基本一致，编写程序的代码类似

不同：

异步存取策略：Winsock实现了消息异步存取，socket只有阻塞和非阻塞

错误处理方式：Linux错误代码存放在errno和h\_errno两个全局变量中，Winsock错误码由WSAGetLastError函数获得

宏：在Linux系统中，没有为应用程序定义专门的宏，Winsock为了方便用户使用在Winsock.h头文件中定了的宏