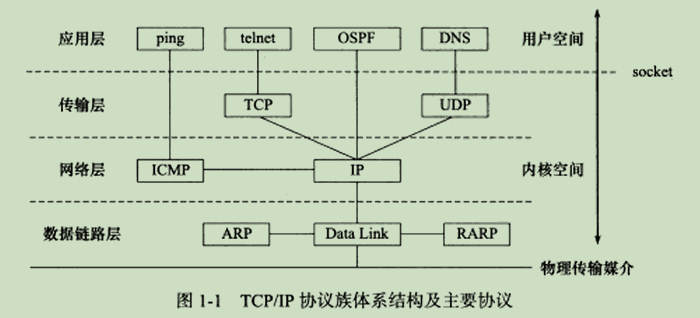
1. TCP/IP协议族

\*\*\*\*



\*数据链路层

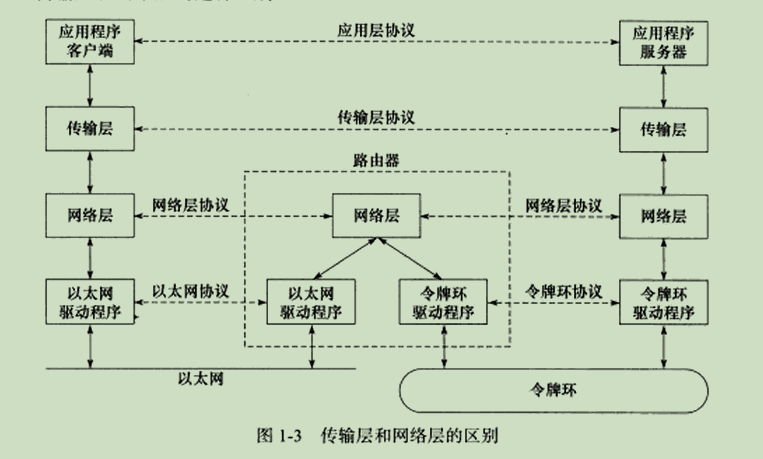
+ARP：使用物理地址寻址一台机器，目标机器IP转化成其物理地址

\*网络层

+IP协议：根据数据包的目标IP地址来决定如何投递它

+ICMP协议：回应网络错误，1.目标不可达（类型值为3）， 重定向（类型值为5），循环冗余校验（CRC），ping程序使用ICMP协议

\*



\*传输层协议有：TCP，UDP，SCTP（传输电话信号）

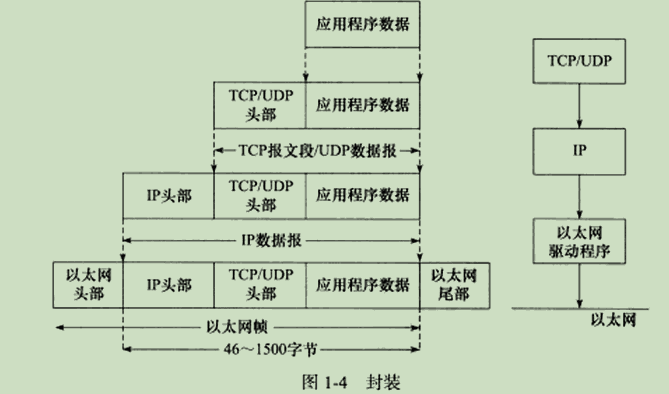
\*应用层协议：

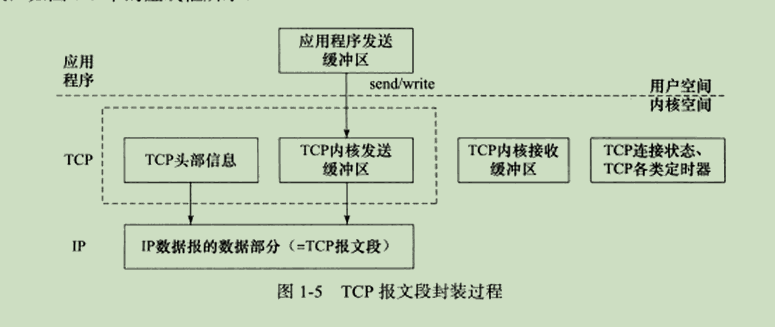
+telnet协议

+OSPF(Open Shortest Path First，开放最短路径优先)协议是一种动态路由更新协议，用于路由器之间的通信，才告知对方各自的路由信息。

+DNS协议

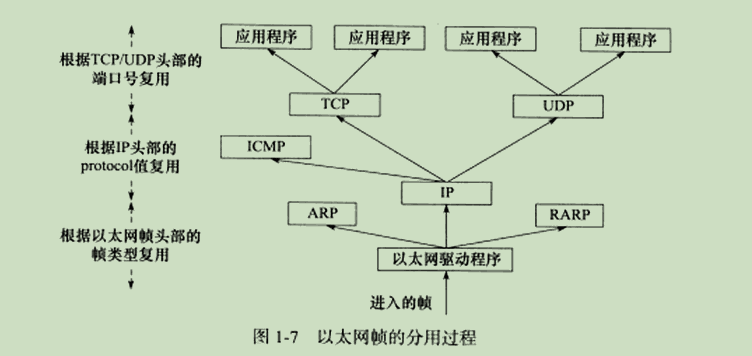
\*





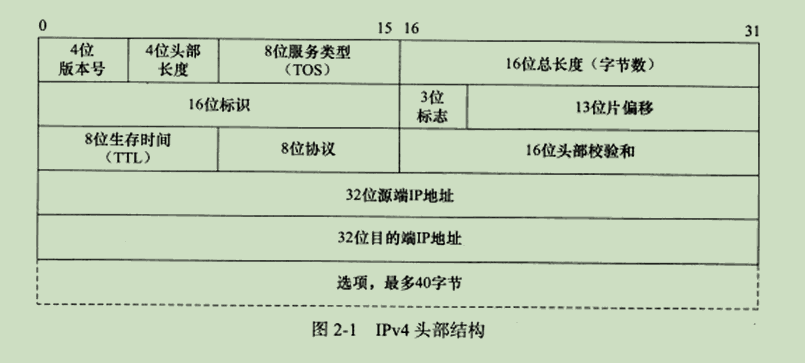
\*MTU：帧的最大传输单元， 以太网的MTU是1500字节。

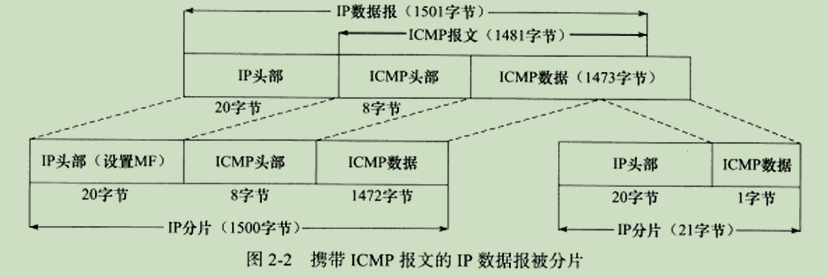
\*

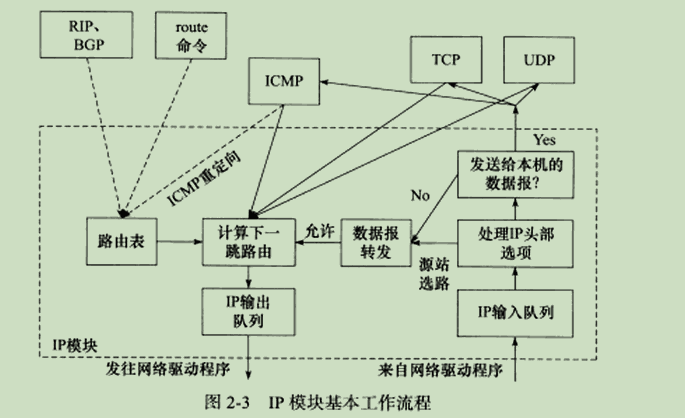


1. IP协议详解

\*



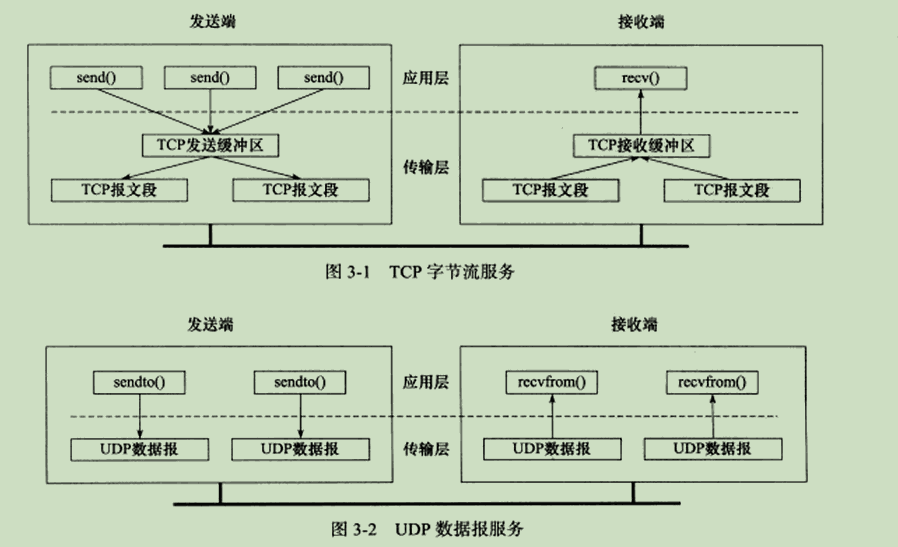




1. TCP协议详解

\*TCP协议连接是一对一，UDP适合于广播和多播

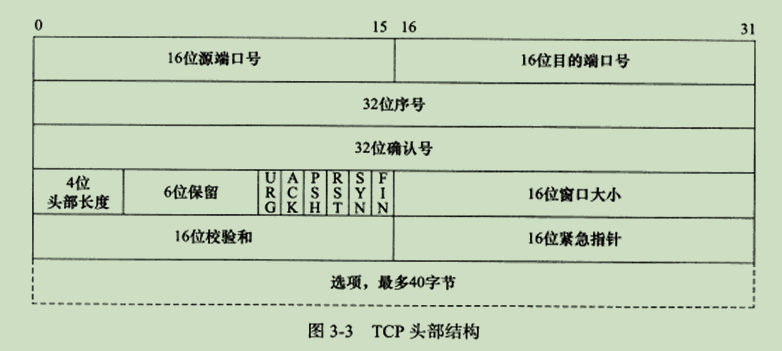
\*



\*TCP可靠：

+1. 采用发送应答机制， 2.超时重传（发送后启动定时器，未收到应答，将重发报文段）

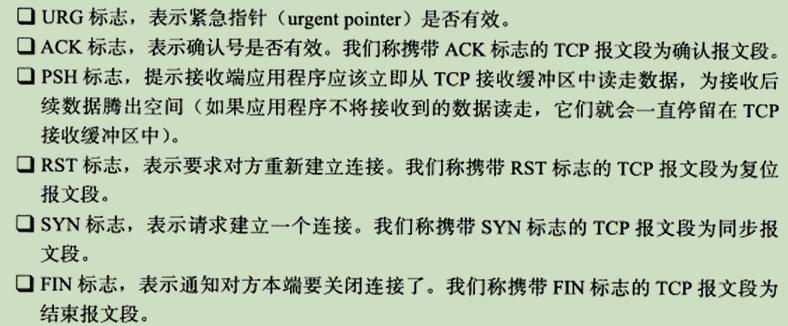
\*



+32位序号：字节流编号（初始化位随机值，后续值为第一个字节流在整个字节流中的偏移）

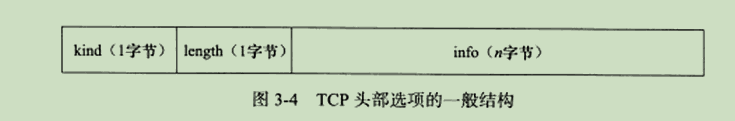
+32位确认号：收到的TCP报文段的序号值加1，

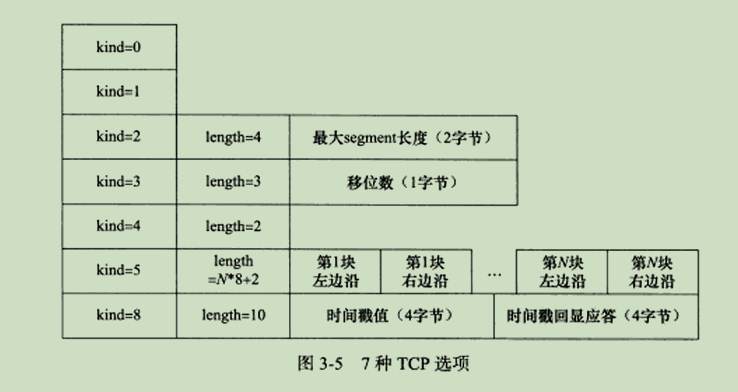
+6位标志位

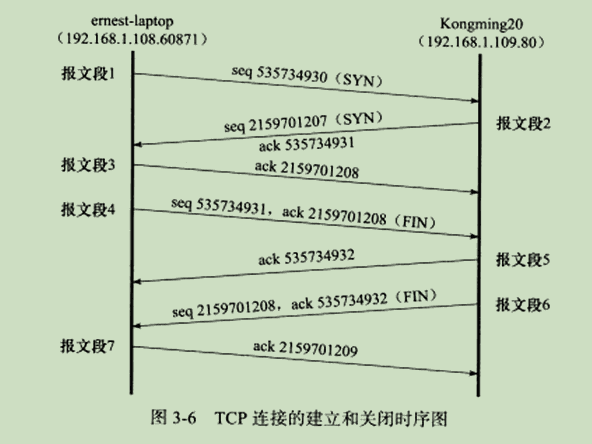


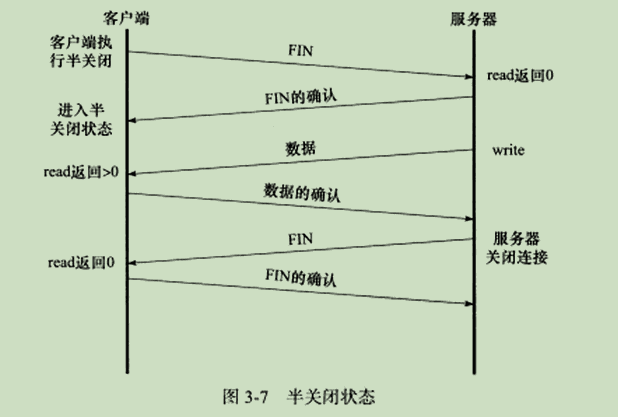
+16位窗口大小：TCP流量控制手段。接收通告窗口（Receiver Window, RWND）,缓冲区大小

+选项

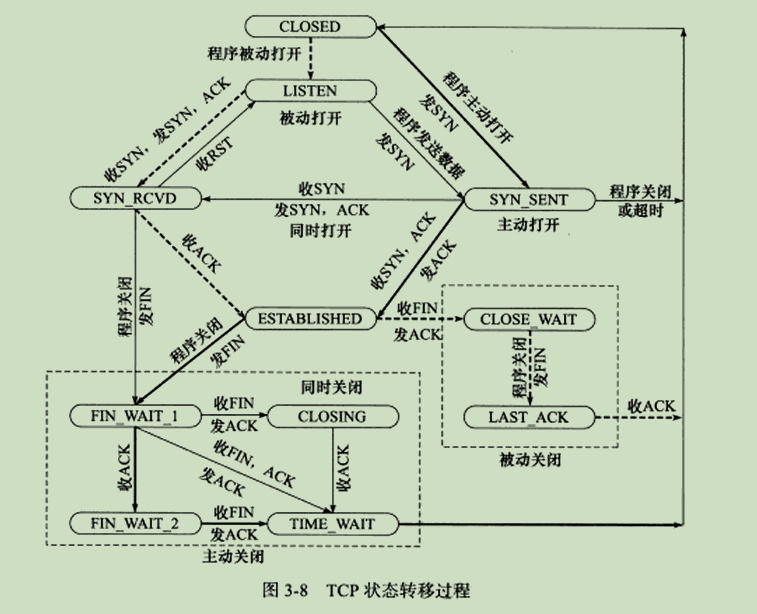


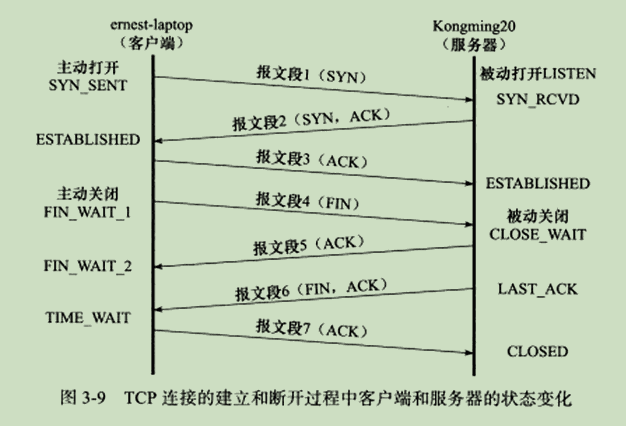






\*





\*产生复位报文段的3种情况

+1.访问不存在的端口，2.异常终止连接， 3.处理半打开连接

\*Nagle算法

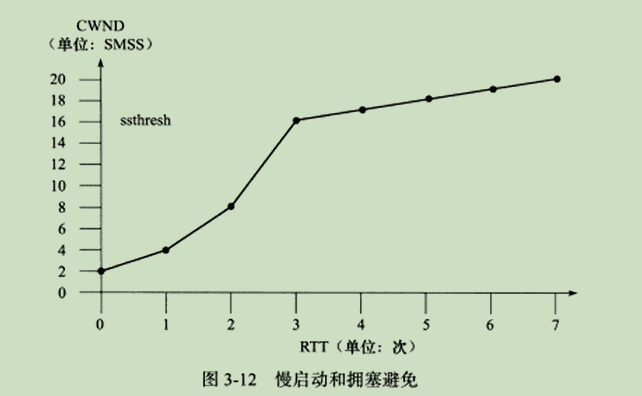
\*TCP拥塞控制的4个部分： (linux有多种算法：reno算法，vegas算法，cubic算法)

+1.慢启动

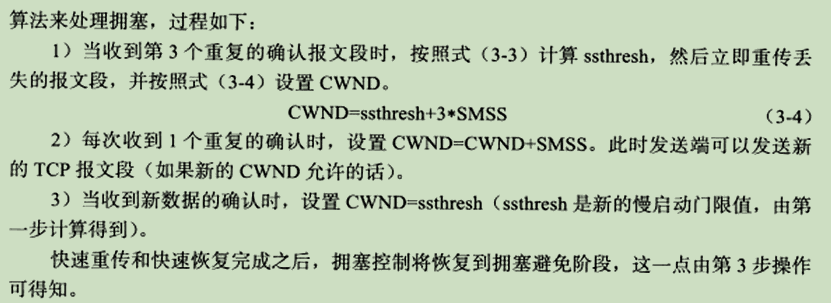
+2.拥塞避免

+3.快速重传

+4.快速恢复



\*快速重传和快速恢复算法来处理拥塞：



1. tcp/ip通信案例： （略）

\*

1. linux网络编程基础api

\*

1. 高级I/O函数

\*pipe函数：创建管道，以实现进程间通信

+管道是单向的。如果需要双向，则需要创建两个管道。

\*dup, dup2:

\*readv,writev函数：

+readv将数据从文件描述符读到分散的内存块中，即分散读。

+writev则将多块分散的内存数据一并写入文件描述符，即集中写。

\*sendfile函数: 两个文件描述符之间直接传递数据，避免内核缓冲区和用户缓冲区直接拷贝数据，效率很高，被称为零拷贝。

\*mmap,munmap函数：

+mmap函数用于申请一段内存空间。可以用来做进程间通信的共享内存，也可以将文件直接映射到其中。

\*splice函数： 用于两个文件描述符之间移动数据，也是零拷贝。

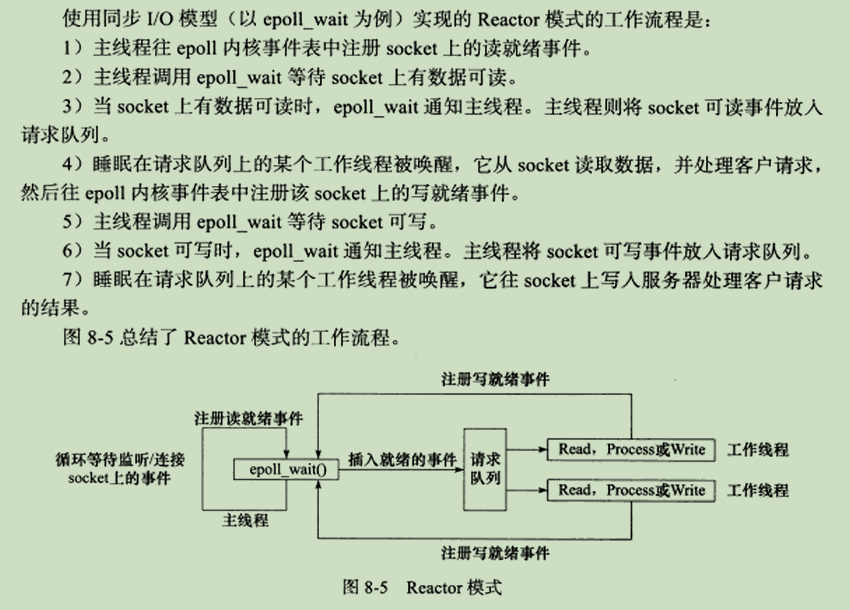
\*tee函数：两个管道文件描述符之间赋值数据，也是零拷贝。

1. Linux服务器程序规范

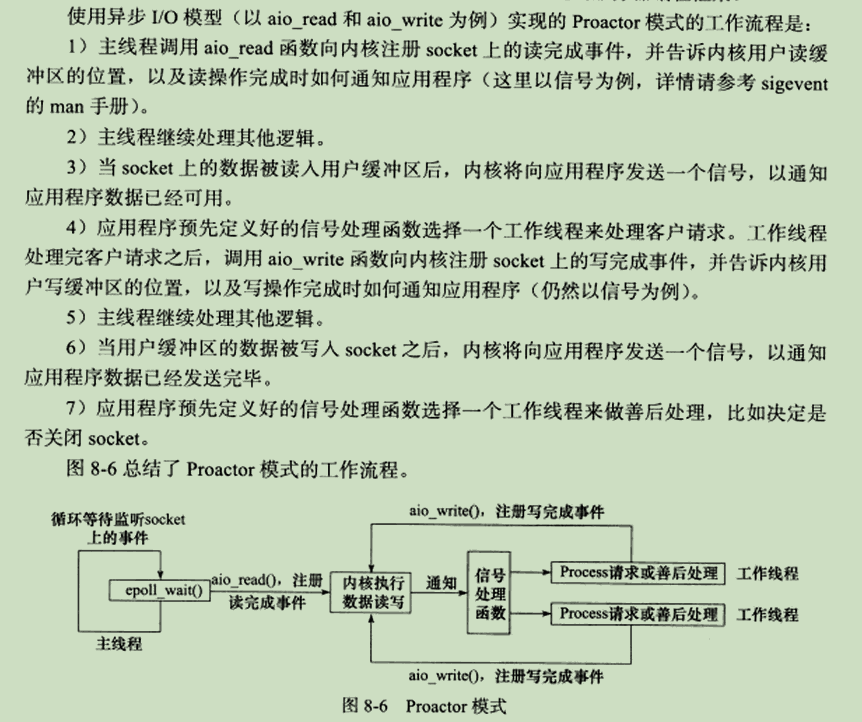
\*

1. 高性能服务器程序框架

\*reactor 模式：主线程（I/O处理单元）只负责监听文件描述符上是否有事件发生，有的话就立即将该事件通知工作线程（逻辑单元）。除此之外，主线程不做任何其他实质性的工作。读写数据，接受新的连接，以及处理客户请求均在工作线程完成。



\*Proactor模式



1. I/O复用

\*epoll

+LT(Level Trigger, 电平触发)：一直触发到该事件被处理。

+ET(Edge Trigger, 边沿触发)：仅触发一次，必须及时处理。

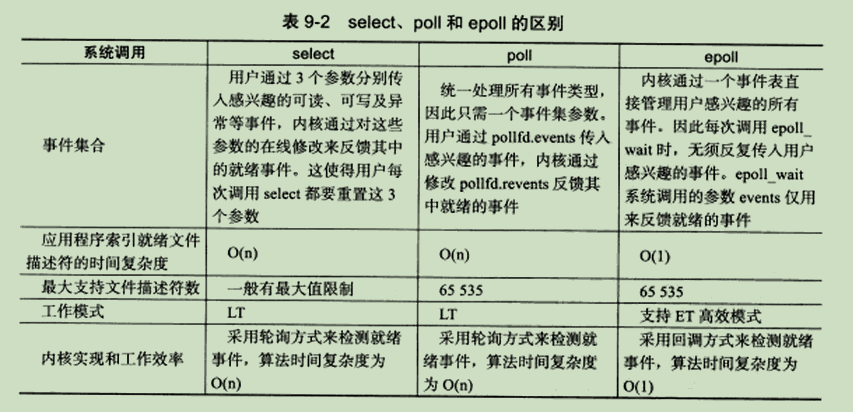
\*三组I/O复用函数的比较

+select: 需要提供3种类型的参数来区分输入，输出，异常等事件。

+poll: 把文件描述符和事件都定义其中，任何事件都被统一处理。

+epoll: 内核维护事件表，无需重复从用户空间读入这些事件。

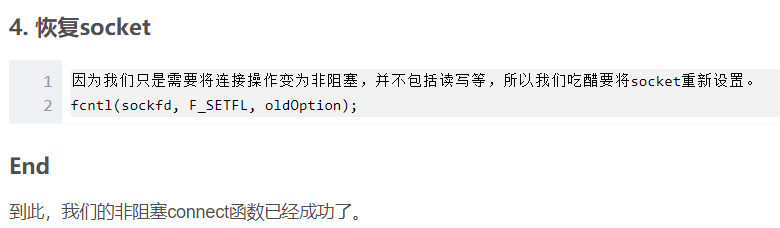
+select，poll使用轮询的方式， epoll使用回调的方式。



\*异步connect实现







\*阻塞，非阻塞，同步，异步 区别解析

<https://cloud.tencent.com/developer/article/1344704?from=information.detail.linux%20%E5%BC%82%E6%AD%A5socket>

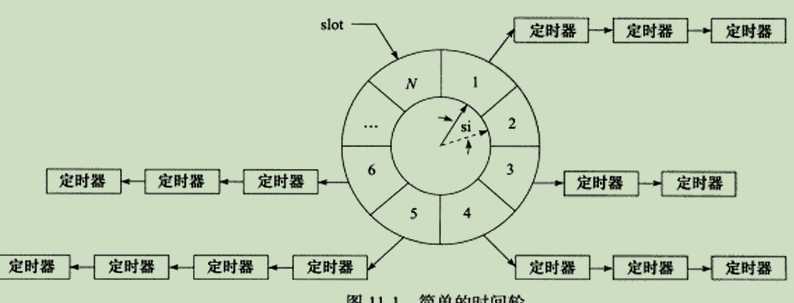
1. 信号

\*

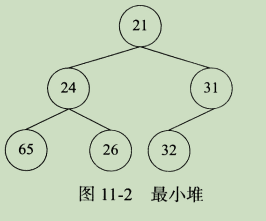
1. 定时器 (需要再研究)

\*升序定时器链表

\*时间轮



\*时间堆



1. 高性能I/O框架库 Libevent (略， 需要专门研究源码)

\*

1. 多进程编程

\*fork

\*exec

\*wait: 阻塞进程，直到该进程的某个子进程结束运行为止。

\*\*\*IPC通信（管道，信号量，共享内存，消息队列，socket， 信号(异步方式)）

\*1.管道：

+无名管道pipe：半双工，数据只能单向流动，父子进程关系才能使用。

+高级管道popen:

+命名管道FIFO：半双工，允许在无亲缘关系进程间通信。

\*2.信号量(Semaphore)：当多个进程同时访问系统上的某个资源的时候，比如同时写一个数据库某条记录，或者同时修改某个文件，就需要考虑进程的同步问题（也用于多线程）。

\*临界区：一段代码引发了进程之间的竟态条件。我们称这段代码为关键代码段，或临界区。

\*信号量API： semget, semop, semctl

\*3.共享内存：最高效的IPC机制，因为不涉及进程之间的任何数据传输。这种高效率带来的问题是，必须用其他辅助手段来同步进程对共享内存的访问。通常和其他进程间通信方式一起使用（如信号量）。

\*共享内存API：shmget, shmat, shmdt, shmctl mmap,munmap

\*4.消息队列：每个数据块都有一个特定的类型，接收方可以根据类型来有选择地接受数据。

\*消息队列API： msgget, msgsnd, msgrcv, msgctl

1. 多线程编程

\*线程同步机制： POSIX信号量，互斥锁，条件变量

\*1. 信号量函数：

+sem\_init: value参数直到信号量的初始值。

+sem\_destroy: 销毁一个正被其他线程等待的信号量，则将导致不可预期的结果。

+sem\_wait:信号量的值减1,。如果信号量为0，则sem\_wait将被阻塞，直到信号量是非0。

+sem\_trywait：sem\_wait的非阻塞版本。信号量为0时，将返回-1 并设置errno为EAGAIN

+sem\_post：信号量值加1。当信号量大于0，则调用sem\_wait唤醒线程。

\*2. 互斥锁：保护关键代码段，有点像一个二进制信号量。

+pthread\_mutex\_init

+pthread\_mutex\_destroy

+pthread\_mutex\_lock

+pthread\_mutex\_trylock

+pthread\_mutex\_unlock

\*3. 条件变量：线程之间同步共享数据的值。当某个共享数据达到某个值的时候，唤醒等待这个共享数据的线程。

+pthread\_cond\_init

+pthread\_cond\_destroy

+pthread\_cond\_broadcast: 以广播的方式唤醒所有等待目标条件变量的线程。

+pthread\_cond\_signal：唤醒一个线程。至于唤醒哪个线程，取决于线程的优先级和调度策略。

+pthread\_cond\_wait: 等待目标条件变量。

1. 进程池和线程池

\*

1. 服务器调制，调试和测试

\*

1. 系统检测工具

\*tcpdump

\*lsof: 列出当前系统打开的文件描述符的工具

\*nc: 快速构建网络连接

\*strace: 测试服务器性能的工具

\*netstat: 网络信息统计

\*vmstat:

\*ifstat: 网络流量监测工具

\*mpstat: 实时监测多处理器系统上每个cpu的使用情况