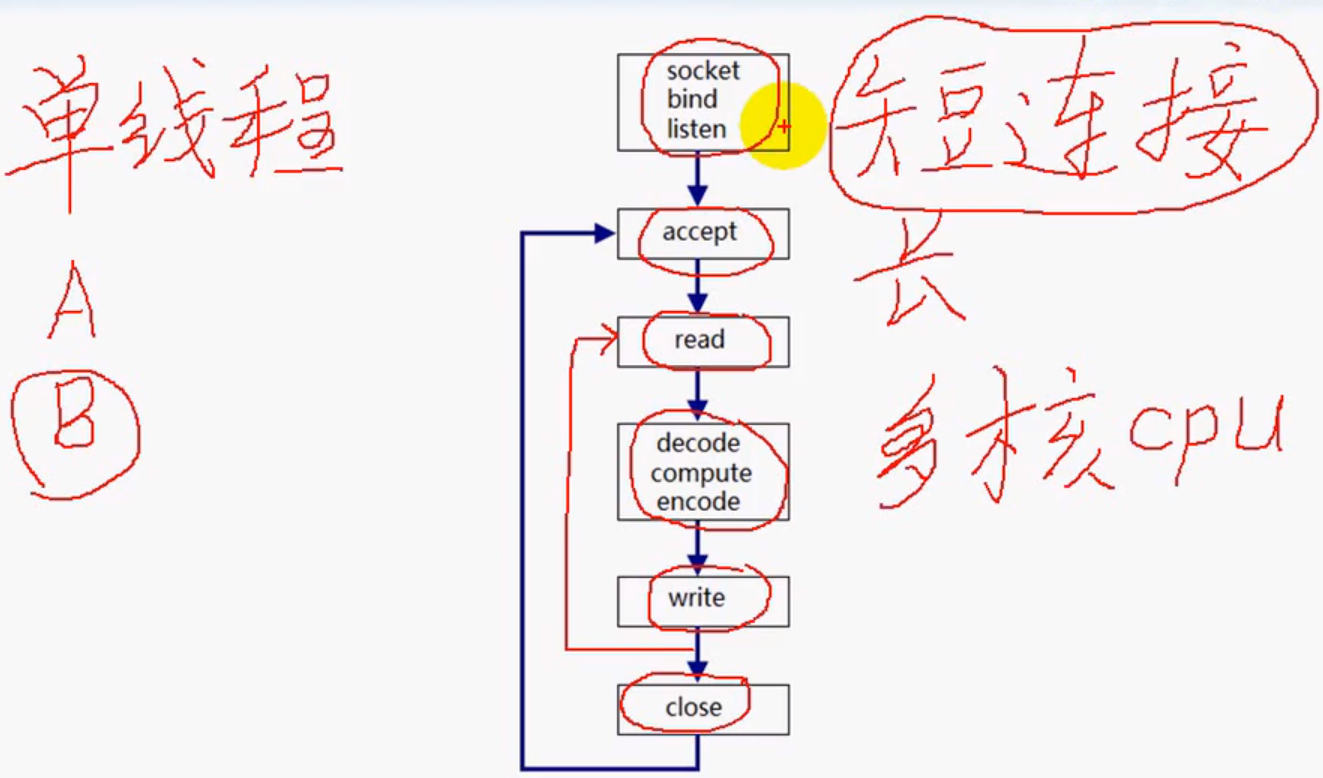
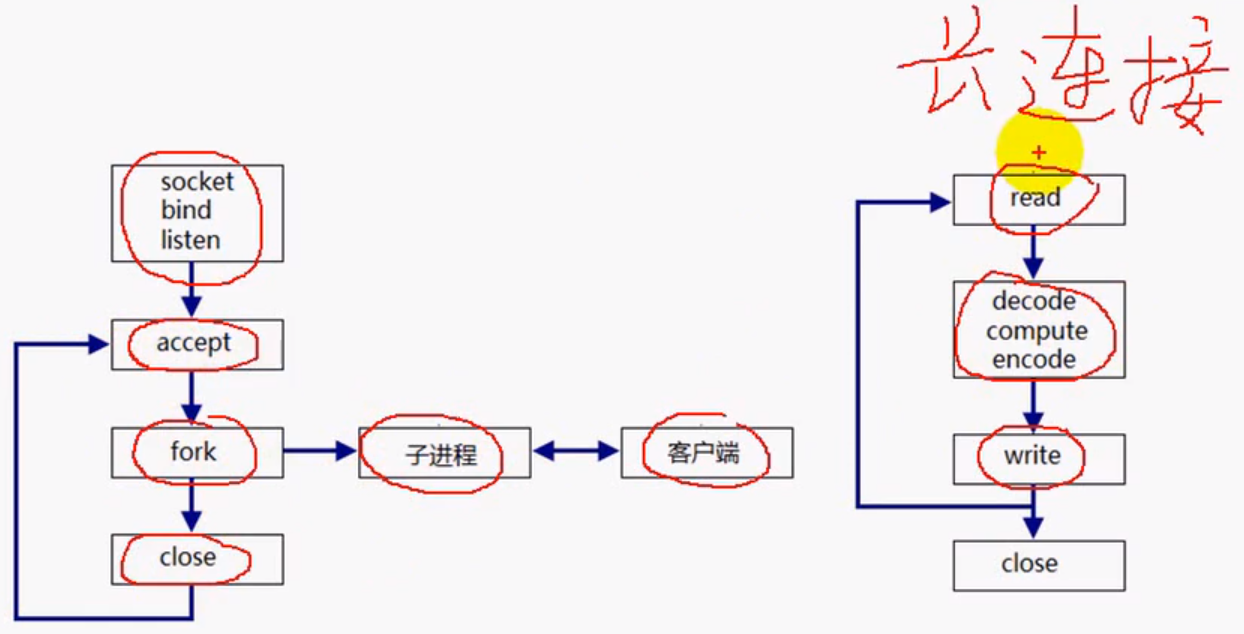
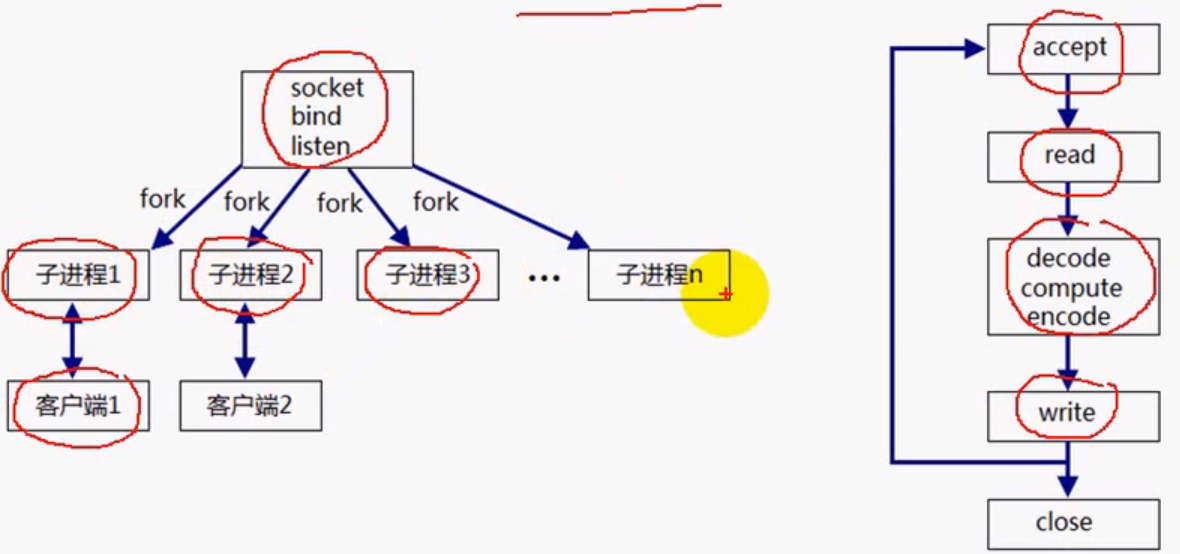
1. **iterative服务器**



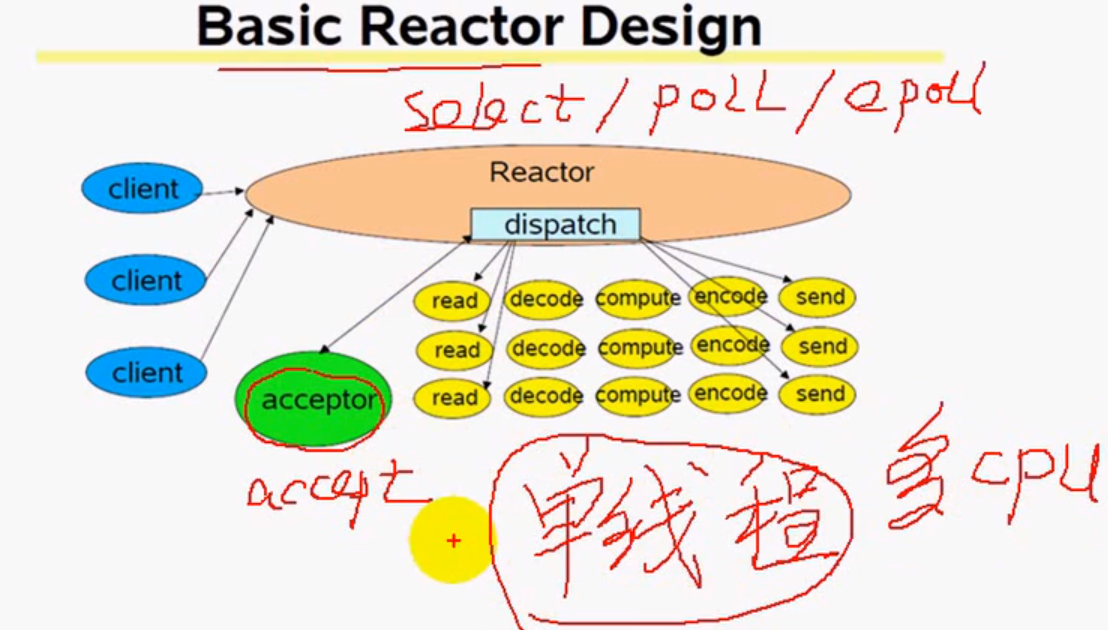
1. 此模型无法充分利用多核CPU。若线程A为长连接，则线程B的响应需等到A连接断开后才能连接。若A为短连接，但是其计算时间较长，则B的响应时间也会增加。
2. **并发式（concurrent）服务器**



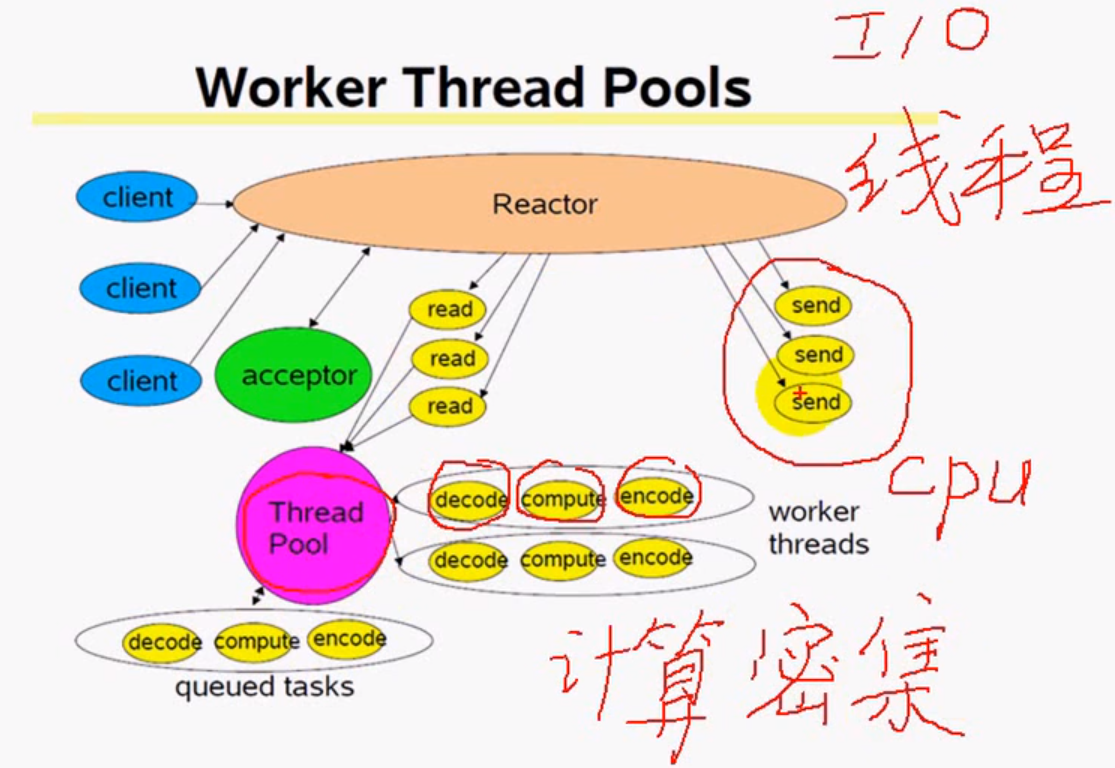
1. 每建立一个连接则创建一个进程/线程，若建立子进程则需要在主进程执行close，因为进程会复制已连接套接字，若建立线程，则不需要执行close，因为线程不会复制已连接套接字。
2. One connection per process/one connection per thread
3. 适合执行时间比较长的服务
4. **perfork服务器**



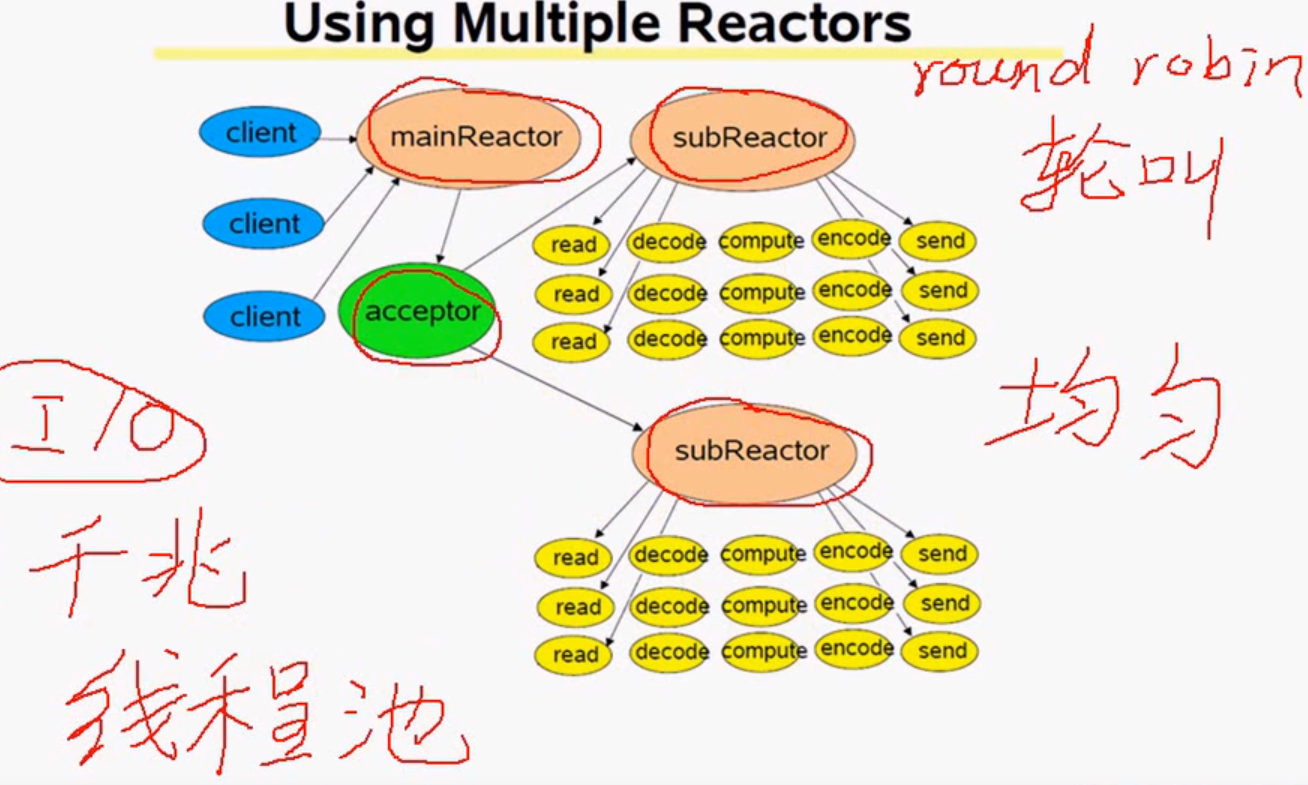
1. 预先建立几个进程/线程
2. 会产生“惊群”现象，因一个accept建立时，多个进程/线程会同时唤醒，但是只有一个会返回正确值；
3. 此模型相比于第二种的优势为减少了创建进程/线程的开销
4. **反应式（reactive）服务器（reactor模式）**



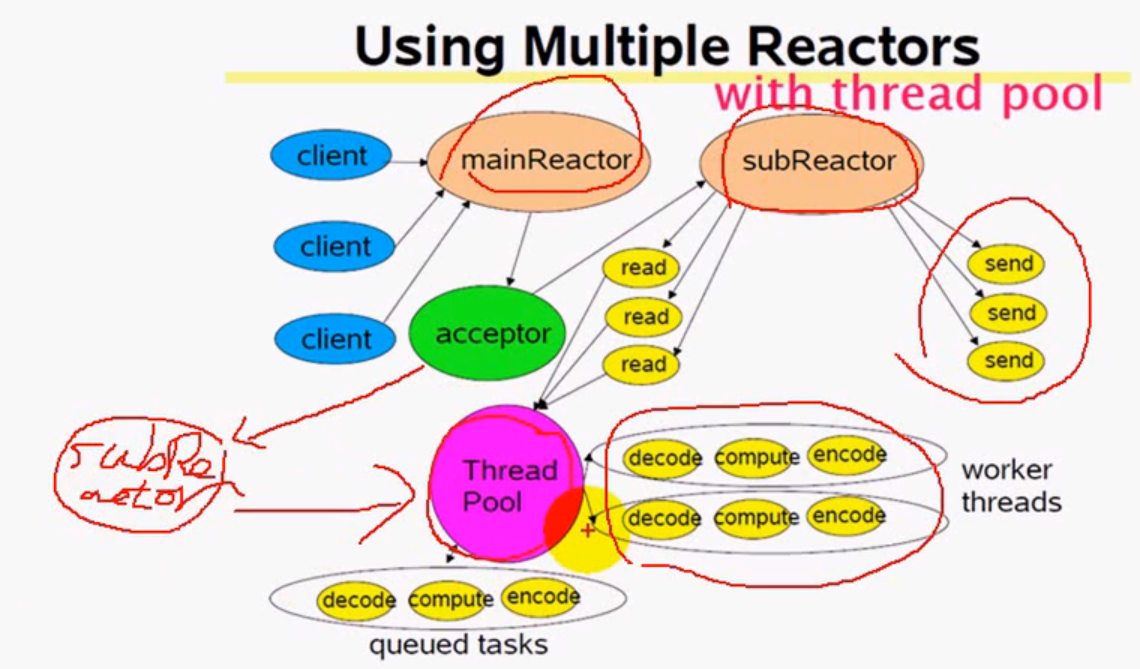
1. 此模型也运行在单线程模式下，无法充分利用多核CPU，但是其相比并发服务器的优势为此模型可以监听的文件描述符更多，应为并发服务器所能创建的进程/线程数量是有限的。
2. 不适合执行时间较长的服务，所以为了让客户感觉是在“并发”处理而不是“循环”处理，每个请求必须在相对较短时间内执行。
3. **reactor+thread per request**
4. 可以利用多核CPU
5. **reactor+worker thread**
6. 每个连接一个线程
7. 但是此模型还不如模型2，因为此模型多了个reactor
8. **reactor+threadpool（适用于计算密集型）**



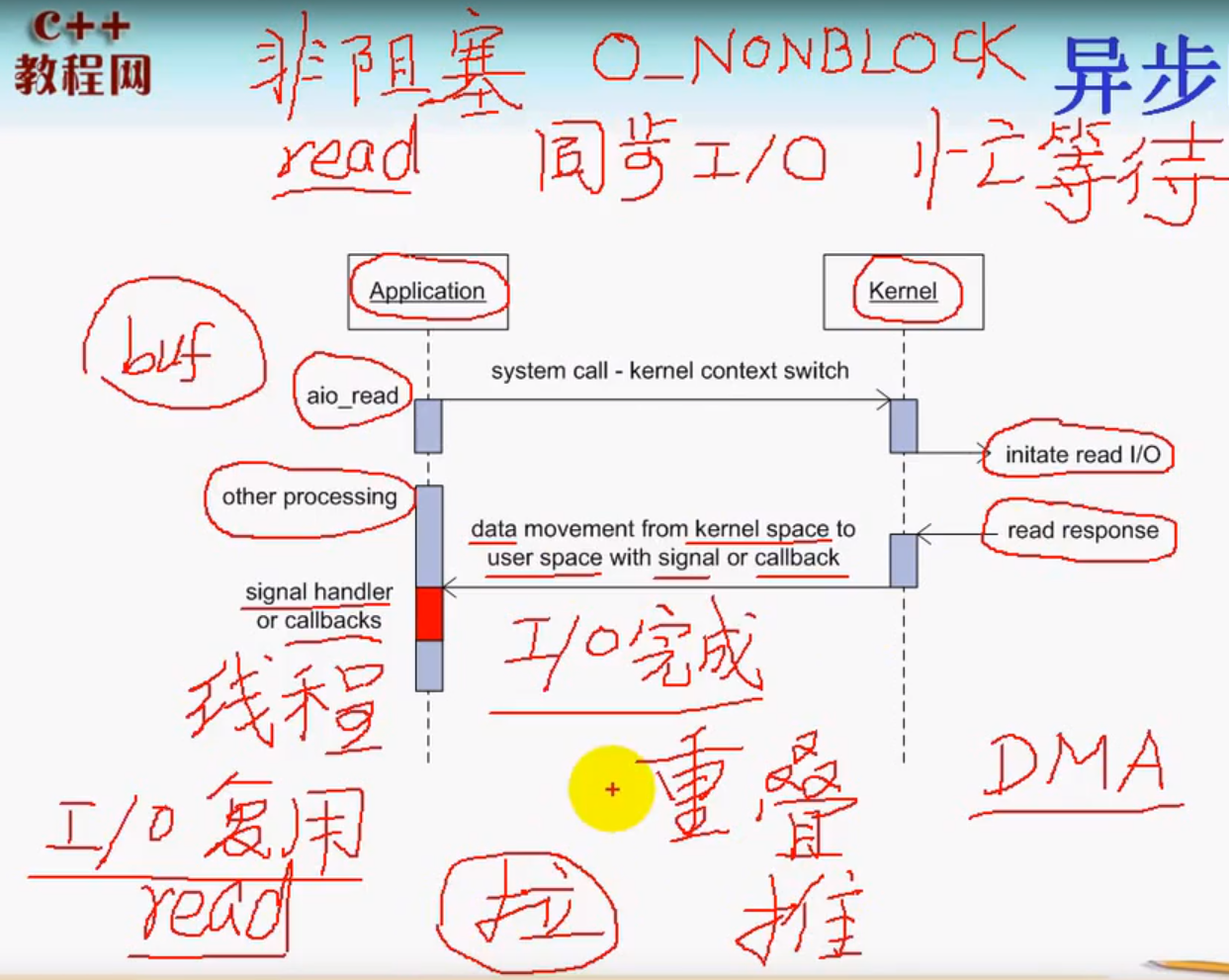
1. 每建立一个连接从线程池提取一个线程，此线程只负责计算，数据的收发还需转到I/O线程
2. **mutiple reactors（能适应更大的突发I/O）**



1. 根据千兆网口的数量确定subreactor的数量；
2. 每建立一个连接，则通过轮询的方式往subreactor中添加连接，这样可以保证连接均匀的分配在subreactor
3. **mutiple reactors+thread pool（one loop per thread+threadpool）（突发IO与密集计算）**



1. 与上个模型相比，将计算部分放到线程池进行。
2. **异步I/O**



1. 同步非阻塞IO：用户态需要不断的调用read函数，此时CPU处于忙等待状态，并且用户态也不执行其他计算操作；
2. IO复用：此时不需要重复调用read函数，因为内核态缓冲区来了数据后通过select/poll/epoll通知用户态，此时用户态再调用read函数将内核态缓冲区的数据复制到用户态缓冲区。用户态也不执行其他计算操作；
3. 异步IO：此时用户态可以执行其他计算操作，当内核态缓冲区来了数据后，会自动将数据复制到用户态，然后应用程序处理。通过信号或注册回调函数方式实现异步IO

**Proactor服务器（proactor模式，异步IO）**

（1）：理论上proactor比reactor效率更高一些

（2）：异步IO能够让IO操作与计算重叠，充分利用DMA特性

（3）：Linux异步IO

Glibc aio

Kernel native aio

1. boost asio实现的proactor，实际上不是真正的异步IO，底层是用epoll实现的，模拟异步IO

11、

1. 多线程能提高并发度么？
2. 如果指的是“并发连接数”，不能
3. 假如单纯采用thread per connection的模型，那么并发连接数大约300（每个线程默认开销为10M），这远远低于基于事件的单线程所能轻松达到的几千甚至几万的并发连接数。所谓基于事件指的是用IO multiplexing event loop的编程模型，又称reactor模式。’
4. 多线程能提高吞吐量么？
5. 对于计算密集型服务，不能
6. 如果要在一个8核的机器上压缩100个1G的文本文件，每个core的处理能力为200MB/s，那么：每次器8个进程，一个进程压缩一个文件与只启动一个进程（8个线程并发压缩一个文件），这两种方式总耗时相当，但是第二种方式能较快拿到第一个压缩完成的文件。
7. 多线程能提高响应时间么？能
8. 多线程如何让IO和计算重叠？
9. 例：日志（logging），多个线程写日志，由于文件操作比较慢，服务线程会等待在IO上，让CPU空闲，增加响应时间
10. 解决方法：单独用一个logging线程负责写磁盘文件，通过BlockingQueue提供对外接口，别的线程要写日志的时候往队列一塞就行，这样服务线程的计算和logging线程的磁盘IO就可以重叠
11. 线程分类？
12. IO线程（特指网络IO），一个reactor就是一个IO
13. 计算线程
14. 第三方库所用的线程，如logging、database