# 标题1

## 标题

## 标题

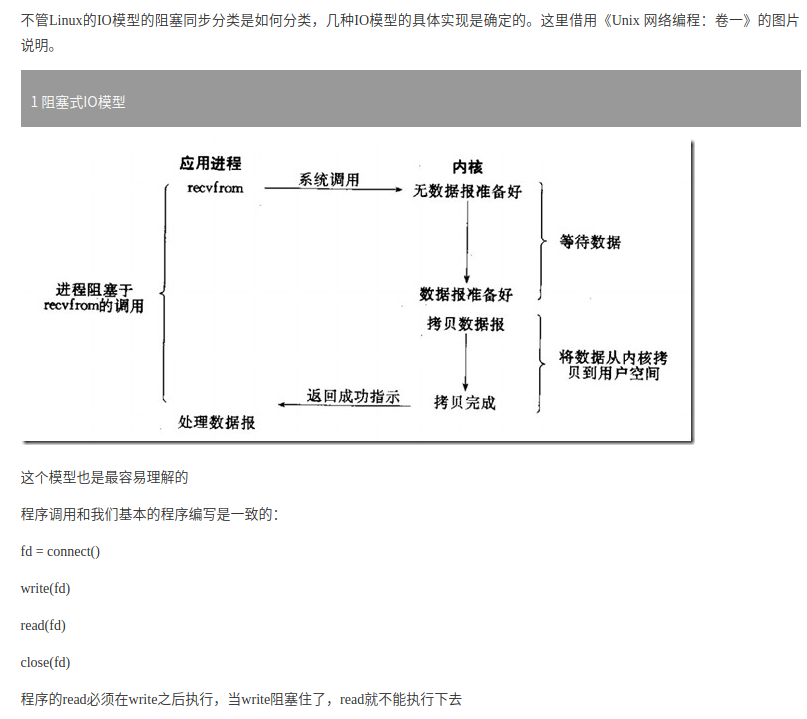
# 网络I/O模型

## 参考资料

<https://blog.csdn.net/lltaoyy/article/details/54861749>

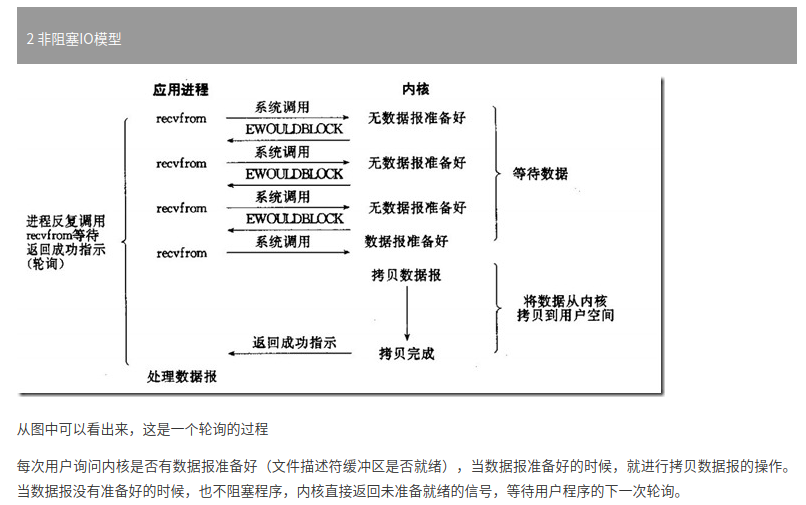
## 阻塞（blocking I/O）

### 定义



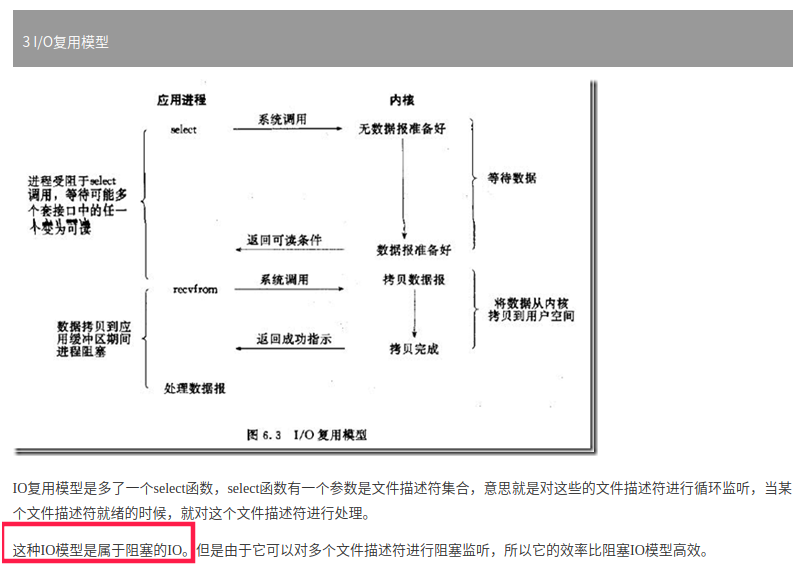
## 非阻塞（non-blocking I/O）

### 定义



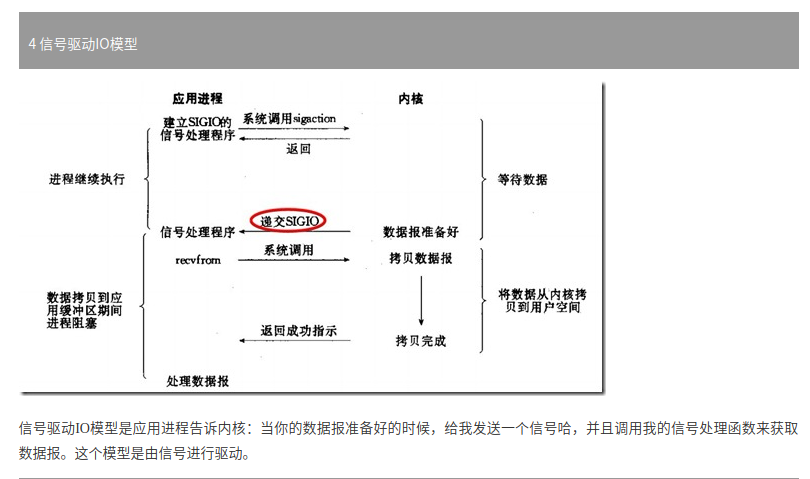
## I/O多路复用（I/O multiplexing）

### 定义



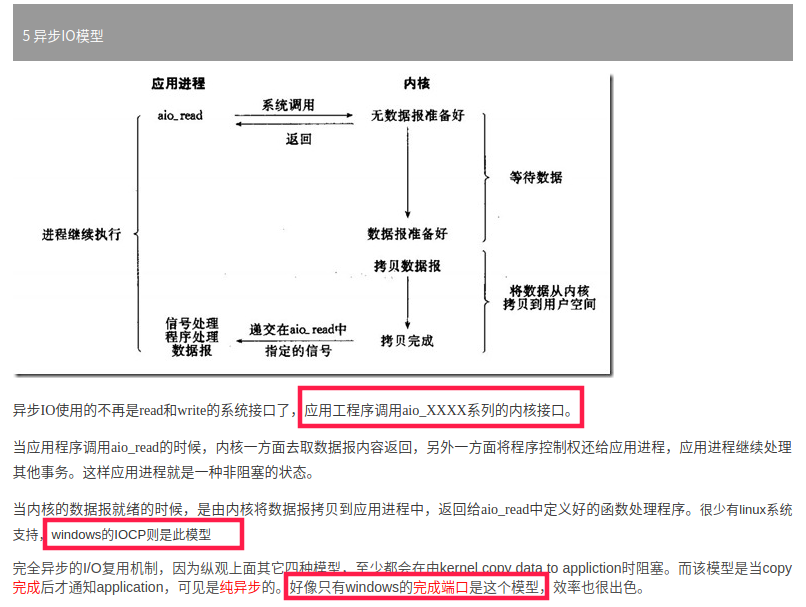
## 信号驱动I/O（signal driven I/O）

### 定义



## 异步I/O（asynchronous I/O）

### 定义



## 重要函数分类

### Select

select对应第3种模型。

### IOCP

IOCP对应第5种模型。

### epoll

那么epoll与kqueue呢？其实也于select属于同一种模型，只是更高级一些，可以看作有了第4种模型的某些特性，如callback机制。

那么，为什么epoll,kqueue比select高级？

答案是，他们无轮询。因为他们用callback取代了。想想看，当套接字比较多的时候，每次select()都要通过遍历FD\_SETSIZE个Socket来完成调度,不管哪个Socket是活跃的,都遍历一遍。这会浪费很多CPU时间。如果能给套接字注册某个回调函数，当他们活跃时，自动完成相关操作，那就避免了轮询，这正是epoll与kqueue做的。

windows or \*nix （IOCP or kqueue/epoll）？

诚然，Windows的IOCP非常出色，目前很少有支持asynchronous I/O的系统，但是由于其系统本身的局限性，大型服务器还是在UNIX下。而且正如上面所述，kqueue/epoll 与 IOCP相比，就是多了一层从内核copy数据到应用层的阻塞，从而不能算作asynchronous I/O类。但是，这层小小的阻塞无足轻重，kqueue与epoll已经做得很优秀了。

# 网络设计模式

## Reactor

### 实例

#### Libevent

#### Epoll

#### Kqueue

#### ACE Reactor

## Proactor

### 实例

#### IOCP

#### ACE Proactor

## Select模型

### 实例

#### Java NIO

# 网络线程模型

## 单线程

### Reactor单线程

#### 只有一个线程

## 多线程

### Reactor多线程

#### 一个Reactor

#### 一个专门线程accept

#### 一组线程处理网络读写

## 主从多线程

### 主从Reactor多线程

#### 多个Reactor

## Master/Worker多线程

### 一个I/O线程

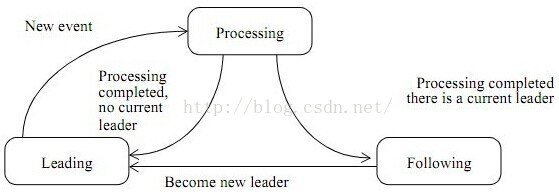
IO线程收到消息后，需要动态分配内存，将请求放在该内存中，然后放入一个队列。

### 多个Worker线程

多个Worker对这个队列进行互斥的操作。

## Leader/Follower多线程

### 状态图



上图就是L/F多线程模型的状态变迁图，共6个关键点：

（1）线程有3种状态：领导leading，处理processing，追随following；

（2）假设共N个线程，其中只有1个leading线程（等待任务），x个processing线程（处理），余下有N-1-x个following线程（空闲）；

（3）有一把锁，谁抢到就是leading；

（4）事件/任务来到时，leading线程会对其进行处理，从而转化为processing状态，处理完成之后，又转变为following；

（5）丢失leading后，following会尝试抢锁，抢到则变为leading，否则保持following；

（6）following不干事，就是抢锁，力图成为leading；

### Leader

永远最多只有一个leader。

所有follower都在等待成为 leader。

线程池启动时会自动产生一个Leader负责等待网络IO事件。

当有一个事件产生时，Leader线程首先通知一个Follower线程将 其提拔为新的Leader，然后自己就去干活了，去处理这个网络事件，处理完毕后加入Follower线程等待队列，等待下次成为Leader。

### Follower

### Processor

### 接口

#### 等待成为Leader（waitToBeLeader）

#### 提升新Leader（promoteNewLeader）

## 半同步/半异步（Half-Sync/Half-Async)

### 同步线程

有一个线程池不断的从队列里获取任务，同步处理，处理完之后再把响应数据由IO线程返回给用户。

### 异步线程

启动一个IO线程使用epoll处理网络数据。当收到一个完整的请求包，把请求放到任务队列。

### 高效半同步/半异步

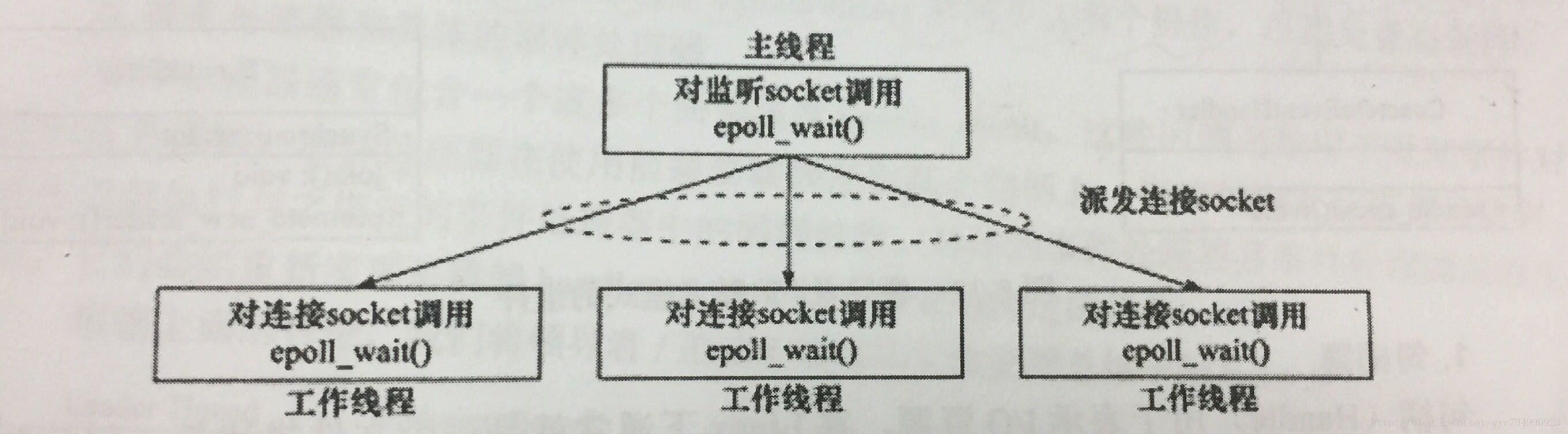
##### 每个工作线程一个I/O Loop

【1】主线程只管理监听socket、连接，得到新的连接socket由工作线程来管理。当有新的连接到来时，主线程就接受并将新返回的连接socket派发给某个工作线程，此后，该socket上任何I/O操作都由被选中的工作线程来处理，知道客户端关闭连接。

【2】主线程向工作线程派发socket的方式，是往它和工作线程之间的管道里写数据。工作线程检测到管道上有数据可读时，就分析是否是一个新的客户端连接请求到来，如果是，就把该socket上的读写事件注册到自己的epoll内核事件表中。

【3】此模式每个线程都维持自己的事件循环，各自监听不同的事件。

【4】



## 半同步/半反应堆（Half Sync/Half Reactor）

# 服务器编程

# 客户端编程

# 重要API

## Linux

### Select

### Epoll

## Windows

### select

### IOCP

## Mac

### kqueue