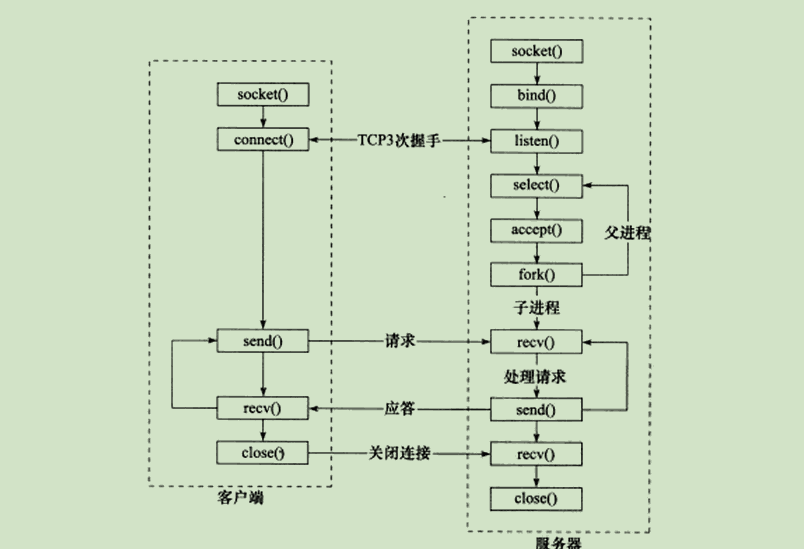
# Linux高性能服务器编程

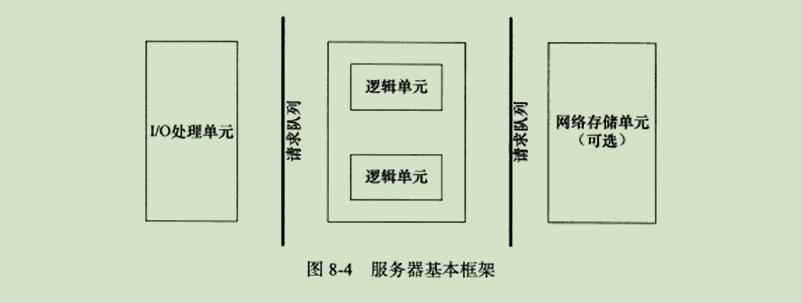
## 高性能服务器框架

### 服务器模型

* B/S，P2P都可以抽象成C/S。
* 服务器和客户端连接的工作流程



### 服务器框架



* I/O 单元：处理连接，读写网络数据，具体实现取决于事件处理模式
* 逻辑单元：codec，业务逻辑处理
* 存储单元：数据库，文件，缓存
* 请求队列：各单元之间的通信方式

### I/O模型

* **同步I/O**

用户自己执行I/O，把数据从内核（socket）缓冲区读写到用户缓冲区，然后处理数据。

阻塞I/O：Socket创建默认是阻塞的（参数，fcntl可以修改），无法立即完成会被操作系统挂起直到事件发生为止。Socket API中可能阻塞的调用（accept，send，recv，connect）

非阻塞IO，总是立即返回，如果事件没有立即发生，系统调用返回-1，需要再次调用，一般要与I/O通知机制（I/O复用和SIGIO信号）一起使用；

I/O复用：读写本身是阻塞的，I/O 复用（poll,epoll）是非阻塞会立即返回。一般放在循环中。

SIGIO信号：为目标FD指定宿主进程，宿主进程捕获信号将触发信号处理函数，报告I/O 事件

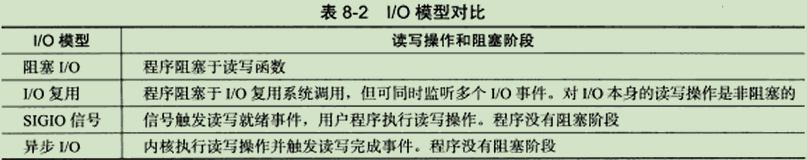
* **异步I/O**

用户直接对I/O执行读写操作，就是告诉内核用户缓冲区的位置之后立马返回，内核再把数据放到用户缓冲区后，像用户报告读写就绪事件。

* **I/O模型对比**

同步：内核向程序通知的是就绪事件，应用程序来完成I/O

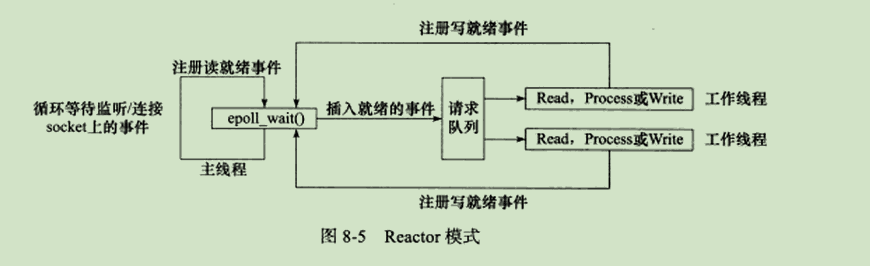
异步：内核向程序通知的是就绪事件，I/O是由内核来完成的

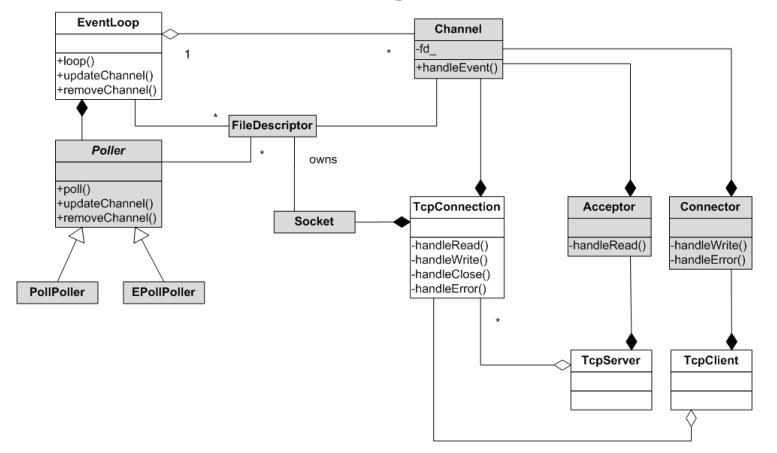


### 高效的事件处理模式

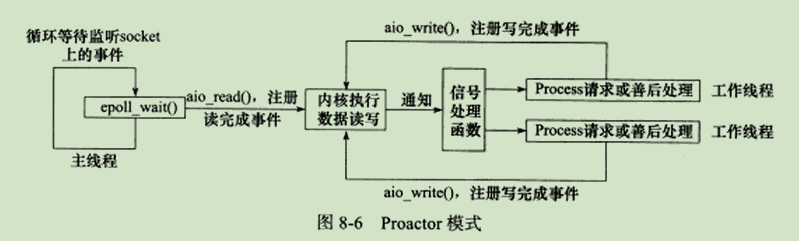
#### Reactor 模式

用于实现同步I/O模型

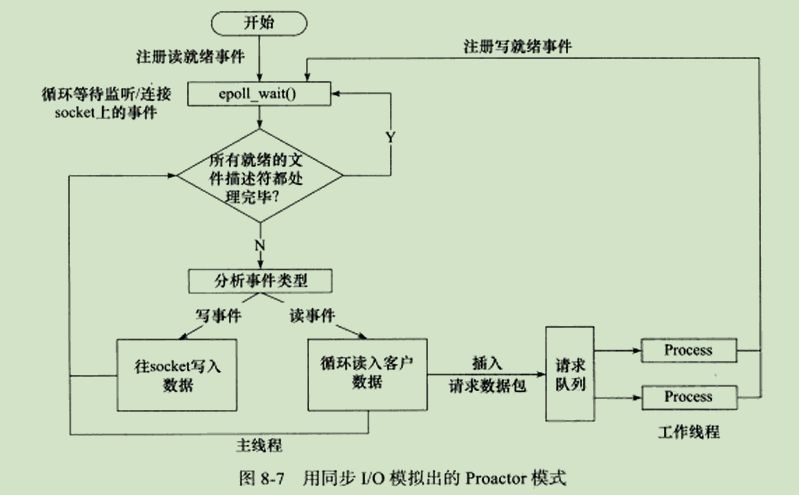




#### Proactor 模式



#### 模拟Proactor 模式



### 两种高效的并发模式

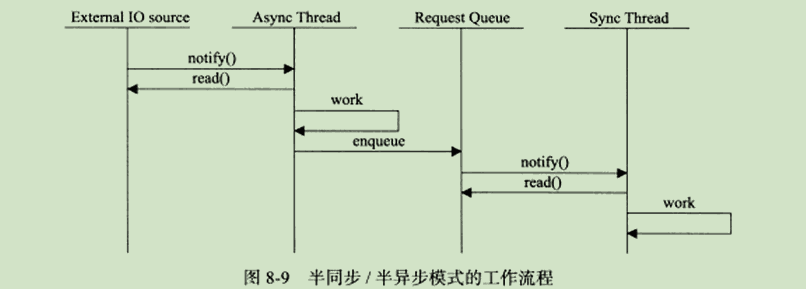
计算密集型：并发没有优势，任务切换效率降低

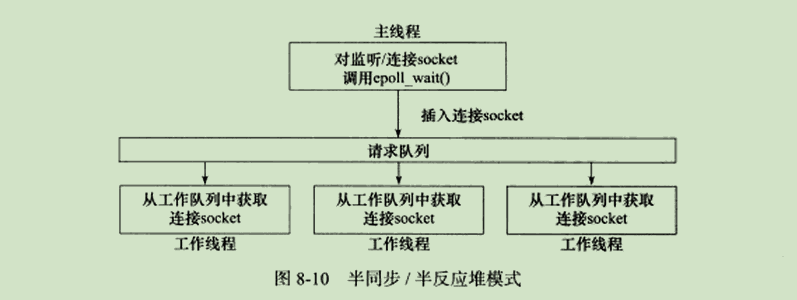
I/O 密集型：I/O速度远低于CPU，阻塞于I/O浪费CPU资源，并发执行提高效率。

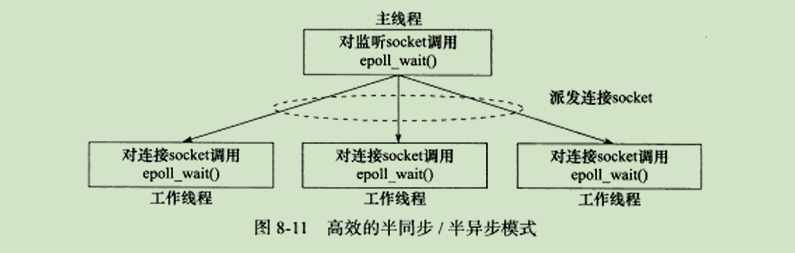
#### 半同步/半异步

同步是按代码顺序执行，效率差逻辑简单。

异步通过事件来驱动，执行效率高，实时性强，相对复杂，难于调试和扩展，不适用于大量并发

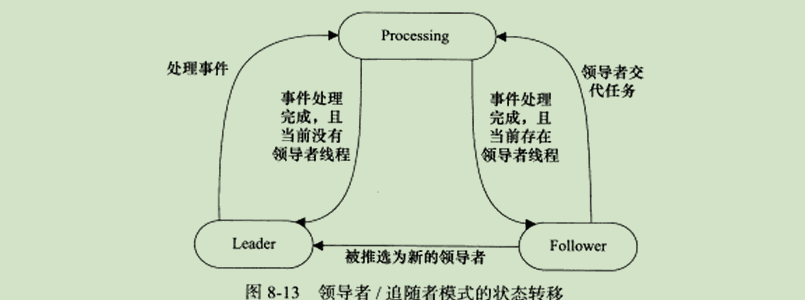




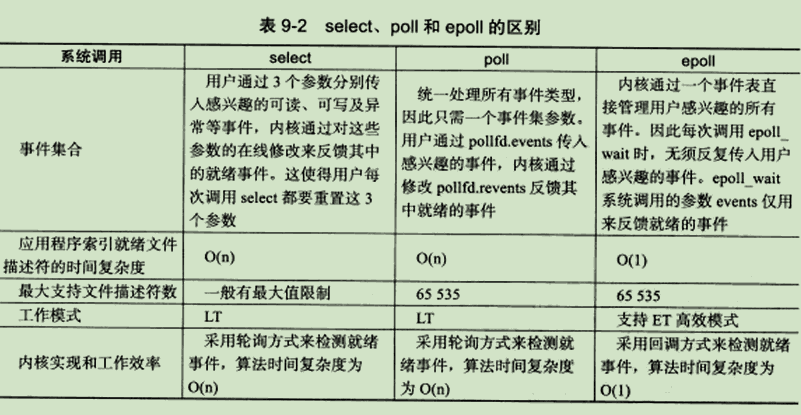


#### 领导者/追随者

多个工作线程轮流获得事件源集合，轮流监听分发并处理事件的一种模式。



## I/O复用



### Select

### Poll

### epoll

## 信号

## I/O框架 libevent