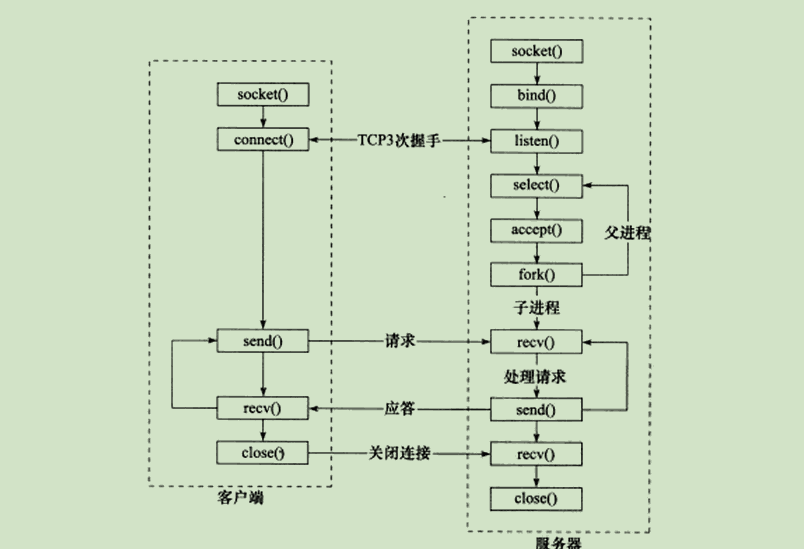
# Linux高性能服务器编程

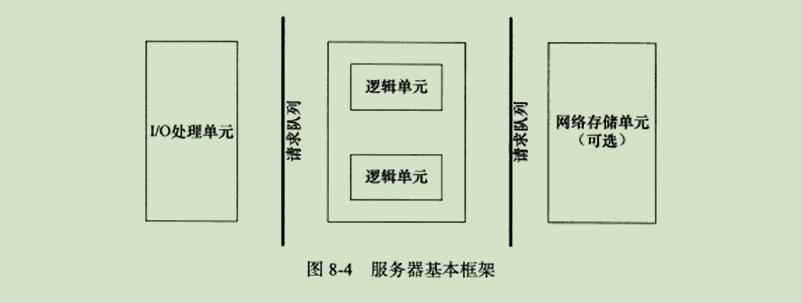
## 高性能服务器框架

### 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；

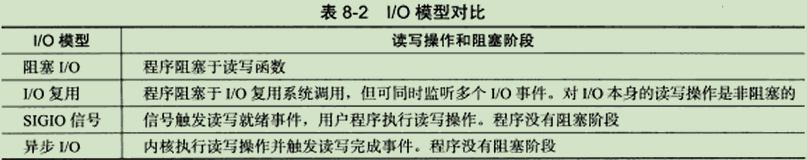
I/O复用：本身是阻塞的，I/O 复用函数（poll,epoll）是非阻塞会立即返回。一般放在循环中。

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**



### 高效的事件处理模式

#### Recator 模式

用于实现同步I/O模型

### 宏定义 和 inline 的区别

Inline和普通函数相比可以加快程序运行的速度，因为编译的时候内联函数直接镶嵌到目标代码当中，不需要中断调用。内联函数会做类型检查，宏不是函数，只是一个简单的替换。

### extern “C” 的作用

C++支持重载，编译后库中的名字于C不同，如：int foo(int x, int y) （\_foo \_foo\_int\_int）。

作用是告诉编译器按C语言的方式编译

## I/O复用

### 指针和引用的区别

## static

* 静态成员函数属于类而不是类对象，所以没有this指针，就是读写静态成员的，不能访问普通成员
* 内存中只有一份数据，所有的对象共享，必须初始化，不能再类的定义中初始化
* 想当于nomember 函数，有利于成为callback，方便作为线程函数

## 虚函数virtual

### virtual 介绍和使用规则

* 虚函数的作用主要是实现了多态的机制，也就是通过基类指针调用实际派生类的成员函数
* 非类的成员函数不能定义为虚函数，
* 类的成员函数中静态成员函数和构造函数也不能定义为虚函数，但可以将析构函数定义为虚函数

### 虚析构

Delete一个指向派生类对象的基类指针，子类和基类的虚构函数都会调用。非虚析构的话，只调用基类的析构函数

### 虚函数表

 为了保证正确取到虚函数的偏移量，编译器必需要保证虚函数表的指针存在于对象实例中最前面的位置

typedef void(\*Fun)(void);

Base b;

Fun pFun = NULL;

cout << "虚函数表地址：" << (int\*)(&b) <<endl; //虚表的地址

cout << "虚函数表 — 第一个函数地址：" << (int\*)\*(int\*)(&b) <<endl;

pFun = (Fun)\*((int\*)\*(int\*)(&b)); // 将地址转化成函数地址

pFun();

(Fun)\*((int\*)\*(int\*)(&b)+0); // Base::f()

(Fun)\*((int\*)\*(int\*)(&b)+1); // Base::g()

(Fun)\*((int\*)\*(int\*)(&b)+2); // Base::h()

## 

## 异常处理exception

## 虚函数virtual

## 虚函数virtual

## 虚函数virtual