# I/O复用

下述情况需要使用I/O复用技术：

1. 客户端需要同时处理多个socket
2. 客户端需要同时处理用户输入和网络连接
3. 服务器需要同时处理监听socket和连接socket
4. 服务器需要同时处理TCP请求和UDP请求
5. 服务器需要同时监听多个端口

注：这里的I/O指的是网络套接字socket的I/O。I/O复用虽然能够监听多个端口，但是其本身是阻塞的，如果需要提高效率需要采用多进程多线程技术。

总结：

# select()

## 原型

#include <sys/select.h>

int select(int nfds,fd\_set\* readfds,fd\_set\* writefds,fd\_set\*exceptfds,struct timeval\* timeout);

参数：

nfds—被监听文件描述符的总数，通常设置为select监听所有文件描述符最大值+1（一般从3开始，0/1/2系统输入/输出/错误）

readfds, writefds, exceptfds—可读/可写/异常文件描述符集合，在调用select的时候，通过这3个参数传入自己感兴趣的文件描述符

timeout—设置select函数的超时时间（select调用失败时返回timeout不确定）

可以访问fd\_set结构体的函数：

FD\_ZERO(fd\_set\* fdset); // 清除fdset的所有位

FD\_SET(int fd,fd\_set\* fdset); // 设置fdset的位fd

FD\_CLR(int fd,fd\_set\* fdset); // 清除fdset的位fd

int FD\_ISSET(int fd,fd\_set\* fdset); // 测试fdset的位fd是否被设置

struct timeval结构体：

struct timeval{

long tv\_sec;//秒数

long tv\_usec;//微秒数

};

如果将两个参数都设置为0，则select立即返回；如果设置为NULL，则select会阻塞。

socket可读的条件：

1. socket中字节大于低水位
2. socket对方关闭连接
3. 监听socket上有新的连接
4. socket上有待处理的错误

## 特点

select()调用的缺点：

1. 单个进程可监视的fd数量被限制；
2. 对socket采用轮询的方法；
3. 需要维护一个用来存放大量fd的数据结构。

# poll()

## 原型

#include <poll.h>

int poll(struct pollfd\* fds,nfds\_t nfds,int timeout);

struct pollfd结构体：

struct pollfd{

int fd; //文件描述符

short events; //注册的事件

short revents;//实际发生的事件

};

## 特点

# epoll()