# WebServer学习笔记：

## V1版本：

### 1.使用了epoll边沿触发+EPOLLONESHOT+非阻塞IO

1.1 EAGAIN-（一般用于非阻塞的系统调用）：

原文链接：<https://blog.csdn.net/liuhengxiao/article/details/46911129>

非阻塞的系统调用，由于资源限制/不满足条件，导致返回值为EAGAIN

例如：首先是把套接字设置为异步的了，然后在使用write发送数据时采取的方式是循环发送大量的数据；由于是异步的，write\send将要发送的数据提交到发送缓冲区后是立即返回的，并不需要对端确认数据已接收。在这种情况下是很有可能出现发送缓冲区被填满，导致write\send无法再向缓冲区提交要发送的数据。因此就产生了Resource temporarily unavailable的错误（资源暂时不可用），EAGAIN 的意思也很明显，就是要你再次尝试。

这个错误经常出现在当应用程序进行一些非阻塞(non-blocking)操作(对文件或socket)的时候。

epoll有两种触发的方式即LT（水平触发）和ET（边缘触发）两种，在前者，只要存在着事件就会不断的触发，直到处理完成，而后者只触发一次相同事件或者说只在从非触发到触发两个状态转换的时候儿才触发。

这会出现下面一种情况，如果是多线程在处理，一个SOCKET事件到来，数据开始解析，这时候这个SOCKET又来了同样一个这样的事件，而你的数据解析尚未完成，那么程序会自动调度另外一个线程或者进程来处理新的事件，这造成一个很严重的问题，不同的线程或者进程在处理同一个SOCKET的事件，这会使程序的健壮性大降低而编程的复杂度大大增加！！即使在ET模式下也有可能出现这种情况！

本文要提到的EPOLLONESHOT这种方法，可以在epoll上注册这个事件，注册这个事件后，如果在处理写成当前的SOCKET后不再重新注册相关事件，那么这个事件就不再响应了或者说触发了。要想重新注册事件则需要调用epoll\_ctl重置文件描述符上的事件，这样前面的socket就不会出现竞态这样就可以通过手动的方式来保证同一SOCKET只能被一个线程处理，不会跨越多个线程。

EINTR错误产生的原因

如果进程在一个慢系统调用(slow system call)中阻塞时，当捕获到某个信号且相应信号处理函数返回时，这个系统调用被中断，调用返回错误，设置errno为EINTR（相应的错误描述为“Interrupted system call”）。例如read,write,open等函数可能发生。

### 2.使用了一个固定线程数的线程池

### 3.实现了一个任务队列，由条件变量触发通知新任务的到来

### 4. 实现了一个小根堆的定时器及时剔除超时请求，使用了STL的优先队列来管理定时器

### 5.mime设计为单例模式

### 6.线程的工作分配为：

主线程负责等待epoll中的事件，并把到来的事件放进任务队列，在每次循环的结束剔除超时请求和被置为删除的时间结点

工作线程阻塞在条件变量的等待中，新任务到来后，某一工作线程会被唤醒，执行具体的IO操作和计算任务，如果需要继续监听，会添加到epoll中

### 7.锁的使用有两处：

第一处是任务队列的添加和取操作，都需要加锁，并配合条件变量，跨越了多个线程。

第二处是定时器结点的添加和删除，需要加锁，主线程和工作线程都要操作定时器队列。

原文链接：<https://blog.csdn.net/daaikuaichuan/article/details/83862311>

多路IO转接服务器也叫做多任务IO服务器。该类服务器实现的主旨思想是，不再由应用程序自己监视客户端连接，取而代之由内核替应用程序监视文件。

主要使用的方法有三种selet，poll，epoll。

select和poll采用轮询方式查询TCP连接上是否有活跃的连接，epoll不这样做，它在Linux内核中申请了一个简易的文件系统，把原先的一个select或poll调用分成了3部分：epoll\_create，epoll\_ctl，epoll\_wait。

1. 调用epoll\_create建立一个epoll对象(在epoll文件系统中给这个句柄分配资源)；

2. 调用epoll\_ctl向epoll对象中添加这100万个连接的套接字；

3. 调用epoll\_wait收集发生事件的连接。

epoll\_event结构体一般用在epoll机制中，其定义如下：

struct epoll\_event

{

  uint32\_t events;   /\* Epoll events \*/

  epoll\_data\_t data;    /\* User data variable \*/

} \_\_attribute\_\_ ((\_\_packed\_\_));

typedef union epoll\_data

{

  void \*ptr;

  int fd;

  uint32\_t u32;

  uint64\_t u64;

} epoll\_data\_t;

epoll\_wait运行的原理是：等侍注册在epfd上的socket fd的事件的发生，如果发生则将发生的sokct fd和事件类型放入到events数组中。并且将注册在epfd上的socket fd的事件类型给清空，所以如果下一个循环你还要关注这个socket fd的话，则需要用epoll\_ctl(epfd,EPOLL\_CTL\_MOD,listenfd,&ev)来重新设置socket fd的事件类型。这时不用EPOLL\_CTL\_ADD,因为socket fd并未清空，只是事件类型清空。这一步非常重要。