

IO多路复用,就像拨开关一样

<https://zhuanlan.zhihu.com/p/159357381>

一段基础的网络编程

先新建socket对象，依次调用bind、listen、accept，最后调用recv接收数据。recv是个阻塞方法，当程序运行到recv时，它会一直等待，直到接收到数据才往下执行。

这里有个非常明显的问题，即在某一时刻，进程收集有事件的连接时，其实这100万连接中的大部分都是没有事件发生的。因此如果每次收集事件时，都把100万连接的套接字传给操作系统(这首先是用户态内存到内核态内存的大量复制)，而由操作系统内核寻找这些连接上有没有未处理的事件，将会是巨大的资源浪费，然后select和poll就是这样做的，因此它们最多只能处理几千个并发连接。而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：

int epoll\_create(int size);

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

int epoll\_wait(int epfd, struct epoll\_event \*events,int maxevents, int timeout);

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。

struct eventpoll {

...

/\*红黑树的根节点，这棵树中存储着所有添加到epoll中的事件， 　　也就是这个epoll监控的事件\*/

struct rb\_root rbr;

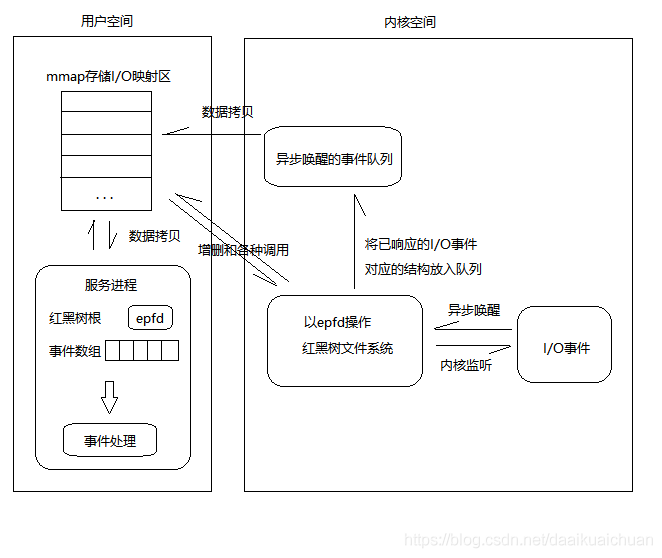
/\*双向链表rdllist保存着将要通过epoll\_wait返回给用户的、满足条件的事件\*/

struct list\_head rdllist;

...

};

我们在调用epoll\_create时，内核除了帮我们在epoll文件系统里建了个file结点，在内核cache里建了个红黑树用于存储以后epoll\_ctl传来的socket外，还会再建立一个rdllist双向链表，用于存储准备就绪的事件



 epoll有EPOLLLT和EPOLLET两种触发模式，LT是默认的模式，ET是“高速”模式。

LT（水平触发）模式下，只要这个文件描述符还有数据可读，每次 epoll\_wait都会返回它的事件，提醒用户程序去操作；

ET（边缘触发）模式下，在它检测到有 I/O 事件时，通过 epoll\_wait 调用会得到有事件通知的文件描述符，对于每一个被通知的文件描述符，如可读，则必须将该文件描述符一直读到空，让 errno 返回 EAGAIN 为止，否则下次的 epoll\_wait 不会返回余下的数据，会丢掉事件。如果ET模式不是非阻塞的，那这个一直读或一直写势必会在最后一次阻塞。

反应堆? 其实就是个事件驱动