* ****select、poll、epoll三组I/O复用系统调用：****
* ****同时监听多个文件描述符****
* ****它们将等待由timeout参数指定的超时时间****，直到一个或者多个文件描述符上有事件发生时返回，返回值就是就绪的文件描述符的数量。返回0表示没有事件发生
* ****3种IO复用介绍分别见：****
* select详情见文章：<https://blog.csdn.net/qq_41453285/article/details/89607091>
* poll详情见文章：<https://blog.csdn.net/qq_41453285/article/details/89672980>
* epoll详情见文章：<https://blog.csdn.net/qq_41453285/article/details/89343074>
* 其中在epoll()的文章中也介绍过三者的异同，详情可以点击上面epoll()的链接
* 下面我们从****事件集、最大支持文件描述符、工作模式、具体实现****等方面介绍它们的异同

# 一、事件集

* 这3组函数都通过某种结构体变量来告诉内核监听哪些文件描述符上的哪些事件，并使用该结构体类型的参数来获取内核处理的结果

## select

* select的参数类型fd\_set****没有将文件描述符和事件绑定****，它仅仅是一个文件描述符集合，因此select需要提供3个这种类型的参数来分别传入和输出可读、可写以及异常等事件
* ****缺点：****
* 一方面，因为select只能处理可读、可写以及异常等事件，所以可读、可写以及异常等事件****不能处理更多类型的时间****
* 另一方面，由于内核对fd\_set集合的在线修改，应用程序在下次调用select前****不得不重置这3个fd\_set集合****
* 每次select都会返回整个用户注册的事件集合（其中包括就绪的和未就绪的），所以应用程序索引就绪文件描述符的****时间复杂度为O(n)****

## poll

* poll对select进行改进，poll的参数类型pollfd****把文件描述符和事件都定义其中****，任何事件都被统一处理，从而使得编程结构简洁得多
* 并且内核每次修改的是pollfd结构体的revents成员，而events成员保持不变，因此下次调用poll时应用程序****无需重置pollfd类型的事件集参数****
* poll与select一样，每次poll都会返回整个用户注册的事件集合（其中包括就绪的和未就绪的），所以应用程序索引就绪文件描述符的****时间复杂度为O(n)****

## epoll

* epoll采用与select和poll完全不同的方式来管理用户注册的事件。它****在内核中维护一个事件表****，并提供了一个独立的系统调用epoll\_ctl来控制往其中添加、删除、修改事件
* 每次epoll\_wait都直接从****内核事件表中取得用户注册的事件****，而****无需反复****从用户空间读入这些事件
* epoll\_wait的events参数仅用来返回就绪的事件，使得应用程序索引就绪文件描述符的****时间复杂度达到O(1)****

# 二、最大支持文件描述符

### ****select****

* 允许监听的最大文件描述符数量通常有限制
* 受到<sys/select.h>头文件中****FD\_SETSIZE宏定义的限制，该宏通常为1024****

### ****poll****

* 用poll系统调用的nfds参数指定最多监听多少个文件描述符和事件
* 系统允许的最大文件描述符数量一般为65535（见/etc/sys/fs/file-max）

### ****epoll****

* 用epoll\_wait系统调用的maxevents参数指定最多监听多少个文件描述符和事件
* 系统允许的最大文件描述符数量一般为65535（见/etc/sys/fs/file-max）

# 三、工作模式

* select和poll都****只能工作在相对低效的LT模式****
* 而****epoll则可以工作在ET高效模式****，并且epoll还支持EPOLLONESHOT事件（该事件可以减少可读、可写和异常等时间被触发的次数）

# 四、实现原理

* select和poll****采用的都是轮询的方式****，即每次调用都要****扫描整个注册文件描述符的集合，****并将其中就绪的文件描述符返回给用户程序，因此检测就绪事件的算法的****时间复杂度为O(n)****
* ****而epoll以红黑树的形式实现,每次epoll\_ctl就是在红黑树中添加删除或者修改节点****。当内核检测到有数据到来时，将触发回调函数，回调函数就将该文件描述符上对应的事件插入内核就绪事件队列。内核最后在适当的时机将该就绪事件队列中的内容拷贝到用户空间。因此epoll\_wait无需轮询整个文件描述符来检测哪些事件已经就绪，其****算法时间复杂度是O(1)****
* epoll\_wait的效率未必比select和poll高，因为此时回调函数被触发得过于频繁。所以****epoll\_wait适用于连接数量多，但活动连接较少的情况****

# 五、总结

