```
信号驱动I/O需用户采用非阻塞的读写方式,并设置O_ASYNC标志。当可执行I/O操作时由内核向相应的进程发送信号
                                                                               gemptyset(&sa);
                                                                             sa.sa_flags = SA_RESTART; /* 当系统调用被中断时,总是重启系统调用 */
                                                                             sa.sa handler = sigioHandler;
                                                                               f (sigaction(SIGIO, &sa, NULL) == -1) { /* 并不关心原来对SIGIO信号的处理,故oldact为NULL
                                                                               perror("sigaction");
                                       为I/O信号SIGIO设置正确的信号处理函数,即sigaction()调用
                                                                                                 fcntl(fd, F_SETOWN, pid);
                                       设定文件描述符的进程属主,即当文件描述符可执行I/O操作时,信号发送给哪个进程或哪些进程
                               基本流程
                                                                                                    flags = fcntl(fd, F_GETFL);
                                       打开O_NONBLOCK(非阻塞)以及O_ASYNC(使用信号驱动I/O)标识,通常使用fcntI()调用一次性完成
                                                                                                     ntl(fd, F SETFL, flags | O NONBLOCK | O ASYNC)
                                       当收到SIGIO信号时,由于信号驱动I/O采用的是边缘触发,所以必须在一次通知时尽可能多地读取数据,直到返回错误为EAGAIN或EWOULDBLOCK为止
                               信号驱动I/O使用O_ASYNC标志主要原因在于历史原因,当时信号驱动I/O也被称之为异步I/O。随着历史的发展,异步I/O由AIO表示
                  信号驱动I/O
                                                           因为更加关心socket的情况,故管道以及FIFO直接忽略
                                                                 监听套接字接收到了新的连接
                               TCP套接字将在何种情况会发出I/O就绪信号
                                                                 connect()函数返回,即主动端进入RSTABLISHED阶段(收到服务端的seg+ack)
                                                                 套接字中有新的输入,或者是发送缓冲区有了足够的空间
                                                                 连接关闭时
                                                在信号一节中有提到:标准信号将不做排队处理,若在进程执行信号处理函数时产生了多个
                                                                                                      所以,需要使用实时信号来代替SIGIO标准信号,使得内核能对其进行排队处理
                                                SIGIO信号,最终只会传递一次,其余的信号将会丢失
                                                   sig >= SIGRTMIN,调用成功后,sig将取代SIGIO被发送:
                               使用实时信号取代SIGIO
                                                   (fcntl(fd, F_SETSIG, sig) == -1) {
                                                   perror("fcntl");
                                                             select、poll仅支持水平触发,且需要通过迭代的方式获取已经就绪
                                                             的文件描述符
                                                                                                       epoll支持水平触发以及边缘触发
                         epoll模型可以认为是select、poll以及信号驱动I/O的升级版
                                                             信号驱动I/O仅支持边缘触发,且需要处理实时信号队列溢出的情
                                                             况,逻辑较为复杂
                                             epoll的核心数据结构称为epoll实例,和一个打开的文件描述符关联,但不会进行I/O操作,而是内核数据结构的
                                             一个句柄
                                                                                成功时将返回一个代表epoll实例的描述符,失败则返回-1
                                                                               参数size并不是一个文件描述符数量的上限,而是告诉内核应该如何为内部数据结
                                                                               构划分初始大小
                                                        原型: int epoll_create(int size);
                                                                               (在内核2.6.8版本以后,该参数实际就是个摆设)
高级I/O模型(2)
                                                                                和open()调用返回的文件描述符一样,epoll_create()返回的描述符
                                                                                用于指代该epoll实例
                                             创建epoll实例
                                                                                 2.6.27内核版本以来提供了新的创建方法,去除了无用的size参数,并且新增了flags参数,用于控制
                                                                                 epoll实例句柄的行为
                                                        原型: int epoll_create1(int flags);
                                                                                 目前仅支持一个flags: EPOLL_CLOEXEC,当进程fork()调用之后执行exec()时,关闭原有进程的epoll实例。对
                                                                                 应于文件的O_CLOEXEC标志
                                                            原型: int epoll_ctl(int epfd, int op, int fd, struct epoll_event *ev); □ 返回0表示成功,-1表示错误
                                                           epfd ⊙ 由epoll_create()或者是epoll_create1()返回的epoll实例句柄
                                                                                            将描述符fd添加至epfd实例中,并将对fd感兴趣的时间附加至ev参
                                                                               EPOLL_CTL_ADD
                                                                                             若文件描述符已经在epfd列表中,则会返回EEXIST错误
                                                                                             修改描述符fd上设定的事件,修改内容同样使用ev参数传递
                                                                                                                                      当成RESTful API来用
                                                           op ⊝ 指定所要执行的操作 ⊖ EPOLL_CTL_MOD
                                                                                             若文件描述符未在epfd列表中,则会返回ENOENT错误
                                                                                             将文件描述符fd从epfd列表中删除,此时会忽略ev参数
                                                                               EPOLL_CTL_DEL
                                                                                            若文件描述符未在epfd列表中,则会返回ENOENT错误
                                             修改epoll的兴趣列表
                                                                文件描述符,可以是套接字、管道、FIFO或者是POSIX信号量等,甚至可以是epoll实例(多层级),但不能是普通文件或
                                                           fd ⑤ 者是目录的文件描述符
                                                                                    ruct epoll_event {
 uint32_t   events; /* 位掩码,用于表示用户感兴趣事件的集合 *
                                                                                     epoll data t data;
                                                                为指向epoll_event的指针 ○
                         epoll编程接口 ⊖ epoll实例
                                                                              同poll()调用一样,epoll采用事件的方式进行注册
                                                                                      EPOLLIN 🖯 可读
                                                                epoll事件(events)
                                                                                      EPOLLOUT 9 可写
                                                            ev
                                                                                      EPOLLET ⑤ 采用边缘触发通知
                                                                                      EPOLLERR © 有错误发生
                                                                                      data值相当重要,因为这是在文件描述符就绪后唯一获取是哪个文
                                                                                      件描述符就绪的信息
                                                                关于文件描述符的信息记录(data)
                                                                                               指针类型 ○ 可以使用一个指向文件结构体的指针
                                                                                              描述符类型 <sup>©</sup> 直接使用fd的值
                                                             由于每个注册到epoll实例上的文件描述符都需要一段无法被交换的内核空间,所以epoll能够检查的文件描述
                                                                                                                             主要与节点内存相关
                                                             符个数是有上限的
                                            对于socket编程而言,该值绝对是够用的
                                                     原型: int epoll_wait(int epfd, struct epoll_event *evlist, int maxevents, int
                                                                                                         成功时返回已就绪的文件描述符个数,0表示超时,-1表示错误
                                                           参数evlist为epoll_event类型所组成的数组指针,用于返回所有已就绪的文件描述符事件,由用
                                                     evlist 🖯 户自行分配空间
                                                     maxevents 🕤 指定evlist数组的最大值
                                                                                        epoll_wait将永久阻塞,直到有文件描述符就绪或者接收信号
                                                                                        如果epoll_wait调用被信号打断,此时返回值可能为-1且errno为
                                                     timeout © 超时时间,同样有3种方案可选
                                                                                 等于0 <sup>⑤</sup> 此时epoll_wait非阻塞的检查,只是看一眼哪些描述符就绪
                                                                                 大于0 🖯 调用至多阻塞timeout毫秒
                                             epoll的相关调用是线程安全的,也就是说,假设在线程A使用epoll_ctl添加了一个文件描述符,那么在线程B
                                            中,该添加将会生效
                                                  从XX_wait()调用就可以看出select、poll以及epoll之间的区别,epoll将直接返回已就绪的文件描述符列表,那么当用户进行迭代操作时,不会进
                                            效率 ○ 行无用的操作,但select、poll则无法保证
                                                    当应用程序使用epoll模型时,对于存在读写的双工文件描述符,如socket进行操作时,程序代码将会出现割裂。加大了开发人员的开发以及
                         epoll的效率以及潜在的问题
                                                    调试难度,其维护成本更高
                                                    epoll模型自内核2.6版本就有,但是现如今基于epoll的网络框架甚少,除去Redis、Nginx等使用原生C语言编写的程序以外,其余语言
                                                    非常少见于基于epoll的网络框架
```