**Epoll实践理解记录**

1. 深入理解
2. 我对于epoll的理解。Epoll是一种IO复用方式，类似的还有select，poll.如果在一个线程，如果你想要同时处理多个IO（如socket套接字，定时器等），就需要IO复用了。Epoll可以说是一种IO复用策略。怎么用高效怎么来。以客户端连接为例，当有多个连接时，每个套接字都向同一个Epoll注册自己关心的事件（读写），使用Epoll\_ctl()，然后对连接的套接字（文件描述符）进行轮询Epoll\_wait（）,轮询的过程可以设置为阻塞，也可以设置为非阻塞。如果有事件被触发，则Epoll\_wait()会返回相应的事件和套接字（文件描述符）。这个时候，再调用read,write等函数根据套接字从内核读取或者写入内容。当然轮询的过程在一个死循环里进行。相比于select，epoll只轮询活跃的连接。

For（；；）

{

Int iret = Epoll\_wait();

{

If(iret>0) 调用相应的处理函数。（read/write/..）;

}

}

或者：

While ture

{

Active\_stream[] = epoll\_wait(epollfd)

For i in active\_stream[]

{

Read or write till

}

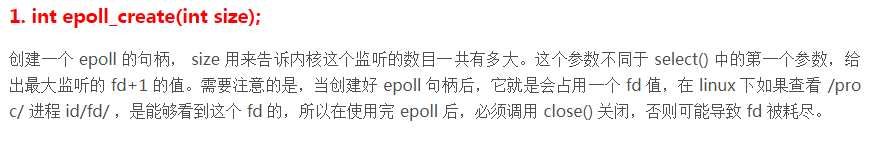
}

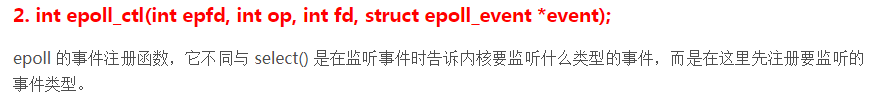
1. 当然，对整个应用来说，是不会直接使用Epoll的。Reactor是一种基于Epoll的事件反应机制。Epoll作为它的核心模块，起着事件分离派发的作用。整个Reactor流程：

定义Reactor变量---->注册事件(实际注册Epoll/IO)----->事件分离派发（Epoll轮询io）------->函数处理（触发事件队列）。

1. 使用Epoll模型，一般都是单线程，或者说一般都是有主线程负责io的轮行，事件全部派发给工作线程。事件驱动。
2. Epoll三部曲。

相关函数介绍(一共就3个函数)





Int epfd：参数是epoll\_create（）的返回值。

Int op: 表示动作。用3个宏表示。

EPOLL\_CTL\_ADD : 注册新的fd到epfd中。

EPOLL\_CTL\_MOD: 修改已经注册的fd的监听事件。

EPOLL\_CTL\_DEL：从epfd中删除一个fd.

Int fd ：需要监听的fd。一般就是套接字，或者其他文件描述符（可以直接是宏定义， 如STDOUT\_FILENO）。

Struct epoll\_event : 告诉内核需要监听什么事，struct epoll\_event 结构如下：

Struct epoll\_event

{

\_uint32\_t events;

epoll\_data\_t data; //这里是用户数据，可以自行填写。

}

Typedef union epoll\_data

{

Void\*ptr;

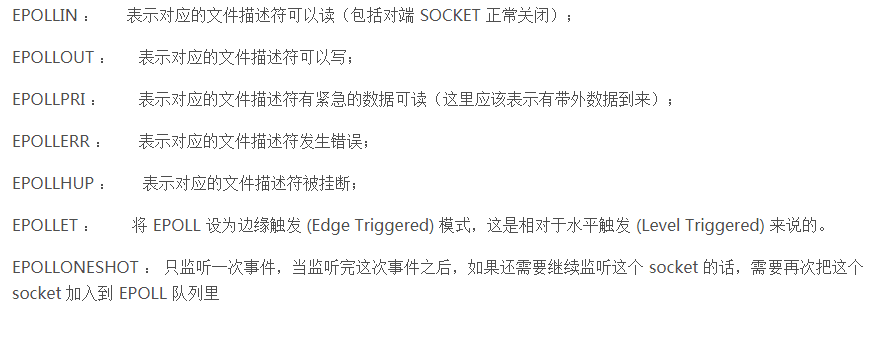
Int fd;

Uint32 u32;

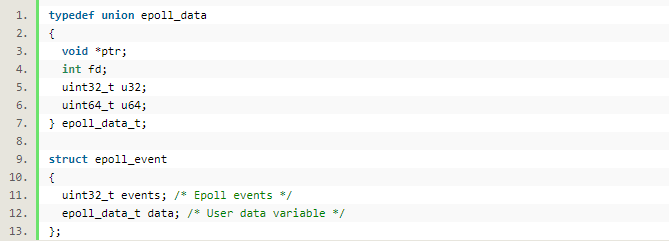
Uint64 u64;

}epoll\_data\_t;

Events 可以是以下几个宏的集合。（用 | 连接多个宏）



联合体epoll\_data\_t 为一个公用体。一般可以使用ptr或者fd记录用户信息，这样在epoll\_wait()响应返回的epoll\_event里，可以根据这个去判断到底是那个文件描述符就绪了。

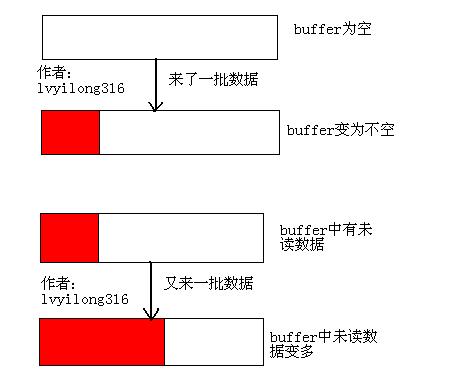


关于 EPOLLHUP ,如果event不指定，则默认为水平触发。（此处的触发，与wait返回就绪有关系）。

ET:(边缘触发)

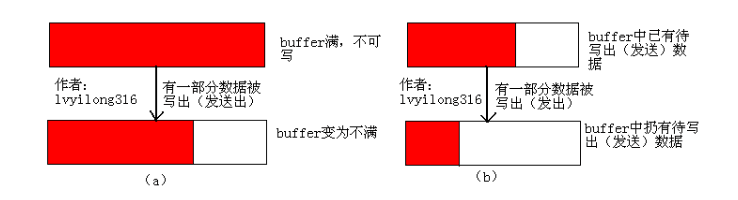
对于读：（①程序 [读入] ②缓存buffer）

1. 当buffer由不可读变成可读的时候，即由空变为非空的时候。（发生在缓存区收到数据）。
2. 当有新数据到达，即buffer中的待读内容变多的时候。（缓存区数据一次没有取完，又有新的数据到来。）



对于写： (②缓存buffer [写出] ③用户)

1. 当buffer由不可写变成可写的时候，即由满状态变为不满状态的时候。（数据发送出去，缓存区出现空间，这时写就绪）。
2. 当有旧数据被送走的时候，即buffer数据变少的时候。



LT:(水平触发)

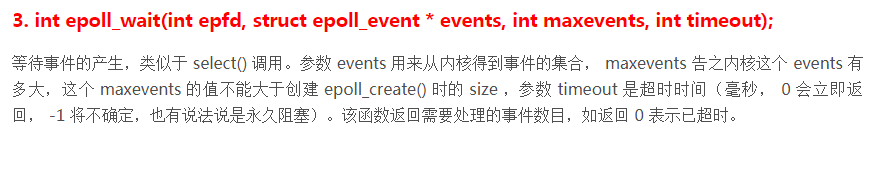
对于读：（程序 读入 缓存buffer，数据从缓存buffer到程序 ）

1. 当buffer中有数据，且数据被读出一部分后buffer还不空的时候，即buffer中的内容减伤的时候，LT模式返回就绪。（只要有数据就触发）。

![9E40_LMLPAJ0FF@9ZH](LDM](data:image/png;base64,)

对于写：（缓存buffer 写出 客户，数据从缓存buffer到客户）

1）当buffer不满，又写了一部分数据后仍然不满。LT模式会返回就绪。



此处的event 便是添加的时候传入的event信息（包括身份信息）。用户可以根据该信息调用相应处理函数，或者回调函数。

1. 编程流程。
2. 编程流程

Epoll\_create()

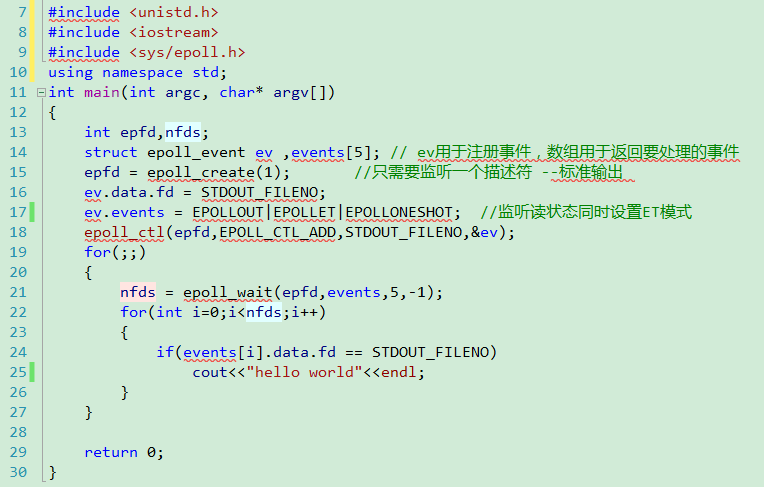
↓

Epoll\_ctl()

↓

Epoll\_wait()

2 .举例分析



2.1 该程序注册了一个标注输出的描述符。每当事件（输出buff为空）就绪，则输出“hello world”。

2.2 第15行 调用epoll\_Create() 创建变量epfd。

2.3 对于18行 epoll\_ctl（）而言，添加了一个标注输出的事件。事件ev的设置为，第16 行拷贝了文件描述符为STDOUT\_FILENO；（如果第16行将将data.fd设置成1000，则第24行data.fd也是1000。以此区分不同事件。）第17行设置监听状态 为可写时触发，并由边缘触发，后面还有只触发一次。（若不加会一直触发）。

2.4 第21行epoll\_wait（）查询就绪的时间。第22-27行。如果就绪的事件使我们需要的， 则输出hello world.(-1会阻塞。改成0则不会阻塞)

2.5 该程序结果是输出一句hello world. 如果去掉 EPOLLONESHOT 则会一直输出 EPOLLONESHOT。

2.6 关于为什么会触发：

首先初始buffer为空，buffer中有空间可写。这时无论是ET还是LT，都会将对应的epitem加入rdlist，导致epoll\_wait()就绪返回写就绪。

程序向标准输出 输出“hello world！”和换行符，因为标准输出为控制台的时候缓冲是“行缓冲”，所以换行符导致buffer中的内容清空，这就对应第二节中ET模式下写就绪的第二种情况。---当有旧数据被发送走时，即buffer中待写的内容变少的时候会触发fd状态改变。所以下次epoll\_wait会返回写就绪。之后重复这个过程，然后循环下去。

3.gdb 调试记录