实验六

实验六
题目6 使用epoll select机制编程
- 实验目的
- 实验内容
- 实验设计原理/步骤
  - select
  - poll
  - epoll
- 实验结果及分析
  - poll与select的区别
  - epoll
- 程序代码，

题目6 使用epoll select机制编程

实验目的

了解掌握Linux系统提供的select、epoll等I/O机制，编写客户端、服务器端程序，编程验证对比这几种网络I/O机制

实验内容

一个基于epoll的dns

一些简单的select，epoll,poll的使用

实验设计原理/步骤

select

> 修改了所给的样例中的select，原本的样例不能写到socket上。

首先初始化应用，设置ip和port，然后，初始化一个 rfds 和 wfds，分别是读的文件标识符的集合和写的文件标识符的集合。

当socket已经建立时，将其加入FD_SET集合中，同时，需要记录maxfd。

之所以记录maxfd，是因为select遍历的数组需要一个遍历的终止值。所以实际上select的效率会和socket的数量成反比。

在 default时，遍历array，当监听文件描述符*listen*描述符，*array[0]*上，发生了rfds的时候，
在代码中是 `FD_ISSET(array[i],&rfds)`
是在 rfds 上接受的时候，进入第一个判断，然后accept这个连接，并生成一个新的socket描述符，之后找到socket数组（array）里，第一个还没有用的项，然后将这个项修改为生成的socket描述符，并修改maxfd。相当于同步的修改了FD_SET中的内容。
当socket连接数目超过array_size后，关闭连接。

当非*listen*描述符，*array[0]* 时，是发生了accept 上的文件读写。
然后 read这个描述符上的所有内容，然后当描述符可写的时候，写入响应值。当没有读到任何信息的时候，就关闭连接，并修改array[j] 为 -1

poll

与select不同的是，select直接给予一个array数组存所有的socket，但是poll是使用一个pollfd结构体，这个结构体的成员变量是fd和events。events需要设置监听的时间，POLLIN，或者POLLOUT，也有POLLPRI，用于urgent data，也有一些其他的宏。

其余和select一样，先判断peerfd[0]的是否有POLLIN 新的连接 ，并进行accept， 并设置evente为POLLIN。

另外，遍历其他accept的连接，对比其中的 `revents 信息`，如果是POLLIN信息，并且读到连接后，将监听的 events修改为POLLOUT，

然后判断 revents是否发生了 POLLOUT的事件，当发生的时候，就写入信息。(可以通过Wget命令测试读取信息。)

epoll

首先epoll 需要使用 epoll_create 创造一个 epoll，用于保存这些文件描述符，然后设置 epoll_event，将这个event添加到epoll中。后续的使用就基本和poll，select等同。

实验结果及分析

epoll，poll，select都是多路复用的IO机制，多路复用的特点就是一个进程能够同时等待多个文件描述符，任意一个进入读就绪状态，select函数就能返回。

select和poll都是效率相对没那么高的机制，select有监视socket描述符的数量限制，并且需要遍历数组才知道谁变化了。

poll与select的区别

是数组换成了链表，同时select用的是位运算，需要分别设置read，write，error的掩码，poll则是设置event参数。另外，select的文件描述符集合 FD_SET 是被内核和用户共同修改的，而poll中，用户修改events，内核修改revents，更简洁。

epoll

著名的libuv库在Linux上就采用的epoll，也就是说nodejs实际上底层用的也是epoll做事件驱动编程。epoll采用回调的形式，poll和select的效率都会随着文件描述符的增加连接数的增加 而下降，但是epoll因为采用回调的形式，所以他的效率和连接总数无关，只有活跃连接数有关。

水平触发：默认工作模式，即当epoll_wait检测到某描述符事件就绪并通知应用程序时，应用程序可以不立即处理该事件；下次调用epoll_wait时，会再次通知此事件。

边缘触发：当epoll_wait检测到某描述符事件就绪并通知应用程序时，应用程序必须立即处理该事件。如果不处理，下次调用epoll_wait时，不会再次通知此事件。（直到你做了某些操作导致该描述符变成未就绪状态了，也就是说边缘触发只在状态由未就绪变为就绪时通知一次）。

程序代码，

在EasyDns中使用epoll的实例

```
    int epollfd = epoll_create(MAX_EPOLL_SIZE);
    struct epoll_event ev;
    struct epoll_event events[MAX_EPOLL_SIZE];
    ev.events = EPOLLIN | EPOLLET;
```

```
    ev.data.fd = listenfd;
    epoll_ctl(epollfd, EPOLL_CTL_ADD, listenfd, &ev);
    while (1)
    {
        int nfds = epoll_wait(epollfd, events, 20, 500);
        for (size_t i = 0; i < nfds; i++)
        {
            if (events[i].events & EPOLLIN)
            {
                int count = recvfrom(listenfd, buf, 1024, 0, (struct sockaddr
*)&clent_addr, &len);
                char *rawmsg = malloc(sizeof(char) * ANS_LEN);
                memcpy(rawmsg, buf, ANS_LEN);
                setblocking(listenfd);
                dealWithPacket(rawmsg, (struct sockaddr *)&clent_addr, listenfd,
count);
                setnonblocking(listenfd);
            }
        }
    }
```