**网络编程学习(七)\_epoll编程实例**

来源：CSDN

一、Epoll模型的三个函数

int epoll\_create(int size);

作用：创建一个epoll句柄，告诉他需要监听的数目(也可以理解成申请一片空间，用于存放监听的套接字)

参数一：通知内核监听size个fd，只是个建议值并与硬件有关系。（从 Linux 内核 2.6.8 版本起，size 这个参数就被忽略了，只要求 size 大于 0 即可）

返回值：返回epoll句柄（fd）

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

作用：控制某个epoll监控的文件描述符上的事件：注册，修改、删除（也就是增添 删除 修改一个事件）

参数一：int epfd:epoll\_create()的返回值

参数二：int op： 表示动作，用三个宏来表示

EPOLL\_CTL\_ADD(注册新的fd到epfd)

EPOLL\_CTL\_MOD(修改已经注册的fd监听事件)

EPOLL\_CTL\_DEL(从epfd删除一个fd)

参数三：int fd 操作对象（socket）

参数四：struct epoll\_evevt\* evevt; 告诉内核需要监听的事件

结构体如下：

struct epoll\_event {

\_\_uint32\_t events; 宏定义读和写EPOLLIN读EPOLLOUT写

epoll\_data\_t data; 联合体

};

联合体如下：

typedef union epoll\_data {

void \*ptr;

int fd;

\_\_uint32\_t u32;

\_\_uint64\_t u64;

} epoll\_data\_t;

返回值：成功返回0,不成功返回-1

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

作用：监听红黑树上的事件，将产生动静的事件放在event这个数组内

参数一：int epfd:epoll\_create()函数返回值

参数二：struct epoll\_events\* events用于回传代处理事件的数组（也就是存放产生动静的事件）

参数三:int maxevents 同时最多产生多少事件，告诉内核events有多大，该值必须大于0

参数四:int timeout表示 -1相当于阻塞，0相当于非阻塞，超时时间(单位：毫秒)

返回值：成功返回产生动静事件的个数

**二、简单demo**

**sever.c**

#include <stdio.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

#include <sys/epoll.h>

#define OPEN\_MAX 1024 //最多连接数

int main(int argc, char \*argv[])

{

int listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

int on = 1;

int i;

int connfd;

setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));//设置为可重复使用的端口

struct sockaddr\_in serv\_addr; //服务器的地址结构体

memset(&serv\_addr,0,sizeof(serv\_addr));

//设置本服务器要监听的地址和端口，这样客户端才能连接到该地址和端口并发送数据

serv\_addr.sin\_family = AF\_INET; //选择协议族为IPV4

serv\_addr.sin\_port = htons(9000); //绑定我们自定义的端口号，客户端程序和我们服务器程序通讯时，就要往这个端口连接和传送数据

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //监听本地所有的IP地址；INADDR\_ANY表示的是一个服务器上所有的网卡（服务器可能不止一个网卡）多个本地ip地址都进行绑定端口号，进行侦听。

bind(listenfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

listen(listenfd, 32);

int efd = epoll\_create(OPEN\_MAX);

struct epoll\_event event, events[OPEN\_MAX];

event.events=EPOLLIN|EPOLLET;

event.data.fd = listenfd;

epoll\_ctl(efd, EPOLL\_CTL\_ADD, listenfd, &event);//把服务器fd包装成事件放在红黑树上

while(1)

{

int nready=epoll\_wait(efd,events,OPEN\_MAX,-1);//判断为可读事件

for(i=0; i<nready; i++)

{

if(!(events[i].events & EPOLLIN))

{

continue;

}

if(events[i].data.fd == listenfd)//表示有新的连接

{

struct sockaddr\_in client\_addr;

int len=sizeof(client\_addr);

memset(&client\_addr,0,sizeof(client\_addr));

connfd = accept(listenfd, (struct sockaddr \*)&client\_addr, &len);

event.events = EPOLLIN|EPOLLET;

event.data.fd = connfd;

epoll\_ctl(efd, EPOLL\_CTL\_ADD, connfd, &event);

}

else//表示旧的数据产生可读事件(1 客户端发来数据 2 客户端断开链接)

{

connfd=events[i].data.fd;

char recvline [1024];

memset(recvline,0,1024);

int nread=read(connfd,recvline,sizeof(recvline));

if(nread==0)

{

printf("client is close..\n"); //打印

epoll\_ctl(efd, EPOLL\_CTL\_DEL, connfd, NULL);//删除果子 select是从集合 和 数组 删除

close(connfd);//关闭客服端 select一样

}

else

{

printf("%s",recvline);

}

}

}

}

return 0;

}

**client.c**

#include <stdio.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in serv\_addr;

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(9000);

if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("inet\_pton failed exit !\n");

exit(1);

}

if(connect(sockfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("connect() failed exit!\n");

exit(1);

}

int len;

char recvline[1024];

char writeline[1024];

while(1)

{

memset(recvline, 0, sizeof(recvline));

memset(writeline, 0, sizeof(writeline));

//发送消息

printf("send to server:");

fgets(writeline,sizeof(writeline),stdin);

write(sockfd,writeline,strlen(writeline));

len = read(sockfd,recvline,1024);

if(len == 0)

{

printf("server is close");

}

printf("receive from server:%s\n",recvline);

}

close(sockfd); //关闭套接字

printf("end exit !\n");

return 0;

}

文章知识点与官方知识档案匹配，

————————————————

版权声明：本文为CSDN博主「weixin\_panda」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_39318565/article/details/109628040