

**计算机学院实验报告**

班 级 网络工程1801 姓 名 吴斌 学 号 18408020129

时 间 2020.12.10 评 分 教师签名

课程名称  **网络程序设计**

1、实验名称； I/O多路复用服务器编程

2、实验目的；

通过在linux操作系统平台下,用C语言进行I/O多路复用服务器网络编程,熟练掌握linux下的I/O多路复用服务器的基本编程方法和select函数以及工作模式。

3、实验原理；

I/O多路复用就通过一种机制，可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。但select，poll，epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的，而异步I/O则无需自己负责进行读写，异步I/O的实现会负责把数据从内核拷贝到用户空间。关于这三种IO多路复用的用法，前面三篇总结写的很清楚，并用服务器回射echo程序进行了测试。

从流程上来看，使用select函数进行IO请求和同步阻塞模型没有太大的区别，甚至还多了添加监视socket，以及调用select函数的额外操作，效率更差。但是，使用select以后最大的优势是用户可以在一个线程内同时处理多个socket的IO请求。用户可以注册多个socket，然后不断地调用select读取被激活的socket，即可达到在同一个线程内同时处理多个IO请求的目的。而在同步阻塞模型中，必须通过多线程的方式才能达到这个目的。

4、实验内容与结果；头文件list.h

#ifndef \_LINUX\_LIST\_H

#define \_LINUX\_LIST\_H

void prefetch(const void \*x){;}

void prefetchw(const void \*x){;}

struct list\_head{

struct list\_head \*next,\*prev;

};

#define LIST\_HEAD\_INIT(name){&(name),&(name)}

#define LIST\_HEAD(name)struct list\_head name=LIST\_HEAD\_INIT(name)

#define INIT\_LIST\_HEAD(ptr)do{(ptr)->next=(ptr);(ptr)->prev=(ptr);}while(0)

static void\_list\_add(struct list\_head \*new,struct list\_head \*prev,struct list\_head \*next){

next->prev=new;

new->next=next;

new->prev=prev;

prev->next=new;

}

static void list\_add(struct list\_head \*new,struct list\_head \*head){

\_list\_add(new,head,head->next);

}

static void list\_add\_tail(struct list\_head \*new,struct list\_head \*head){

\_list\_add(new,head->prev,head);

}

static void \_list\_del(struct list\_head \*prev,struct list\_head \*next){

next->prev=prev;

prev->next=next;

}

static void list\_del(struct list\_head \*entry){

\_list\_del(entry->prev,entry->next);

entry->next=(void \*)0;

entry->prev=(void \*)0;

}

static void list\_del\_init(struct list\_head \*entry){

\_list\_del(entry->prev,entry->next);

INIT\_LIST\_HEAD(entry);

}

static void list\_move(struct list\_head \*list,struct list\_head \*head){

\_list\_del(list->prev,list->next);

list\_add(list,head);

}

static void list\_move\_tail(struct list\_head \*list,struct list\_head \*head){

\_list\_del(list->prev,list->next);

list\_add\_tail(list,head);

}

static int list\_empty(struct list\_head \*head){

return head->next==head;

}

static void \_list\_splice(struct list\_head \*list, struct list\_head \*head){

struct list\_head \*first=list->next;

struct list\_head \*last=list->prev;

struct list\_head \*at=head->next;

first->prev=head;

head->next=first;

last->next=at;

at->prev=last;

}

static void list\_splice(struct list\_head \*list,struct list\_head \*head){

if(!list\_empty(list))

\_list\_splice(list,head);

}

static void list\_splice\_init(struct list\_head \*list,struct list\_head \*head){

if(!list\_empty(list)){

\_list\_splice(list,head);

INIT\_LIST\_HEAD(list);

}

}

#define list\_entry(ptr,type,member)\

((type \*)((char \*)(ptr)-(unsigned long)(&((type \*)0)->member)))

#define list\_for\_each(pos,head)\

for(pos=(head)->next,prefetch(pos->next);pos!=(head);pos=pos->next,prefetch(pos->next))

#define list\_for\_each\_prev(pos,head)\

for(pos=(head)->prev,prefetch(pos->prev);pos!=(head);pos=pos->prev,prefetch(pos->prev))

#define list\_for\_each\_safe(pos,n,head)\

for(pos=(head)->next,n=pos->next;pos!=(head);pos=n,n=pos->next)

#define list\_for\_each\_entry(pos,head,member)\

for(pos=list\_entry((head)->next,typeof(\*pos),member),prefetch(pos->member.next);

&pos->member!=(head);pos=list\_entry(pos->member.next,typeof(\*pos),member),prefetch( pos->member.next ))

#define list\_for\_each\_entry\_safe(pos,n,head,member)\

for(pos=list\_entry((head)->next,typeof(\*pos),member),n=list\_entry(pos->member.next,typeof(\*pos),member);

&pos->member!=(head);pos=n,n=list\_entry(n->member.next,typeof(\*n),member))

#define list\_for\_each\_entry\_continue(pos,head,member)\

for(pos=list\_entry(pos->member.next,typeof(\*pos),member),prefetch(pos->member.next);

&pos->member!=(head);pos=list\_entry(pos->member.next,typeof(\*pos),member),prefetch(pos->member.next))

#endif

源码二selectserver.c

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<errno.h>

#include<netdb.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<arpa/inet.h>

#include<sys/wait.h>

#include<assert.h>

#include"list.h"

#define BUFSIZE 512

struct socket\_opt{

int fd;

int (\*do\_task)(struct sockfd\_opt\*p\_so);

struct list\_head list;

};

typedef struct sockfd\_opt SKOPT,\*P\_SKOPT;

int send\_reply(struct sockfd\_opt\*);

int creat\_conn(struct sockfd\_opt\*);

int init(int);

LIST\_HEAD(fd\_queue);

int num;

fd\_set rfds,orfds;

int maxfd;

static void bail(const char \*on\_what){

fputs(strerror(errno),stderr);

fputs(":",stderr);

fputs(on\_what,stderr);

fputc('\n',stderr);

exit(1);

}

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

int listen\_fd;

struct sockaddr\_in srvaddr;

int port;

int n;

P\_SKOPT p\_so;

socklen\_t optlen;

int sr=1;

if(argc !=2){

fprintf(stderr,"Usage:%s port\a\n",argv[0]);

exit(1);

}

if((port=atoi(argv[1]))<0){

fprintf(stderr,"Usage:%s port\a\n",argv[0]);

exit(1);

}

if((listen\_fd=socket(PF\_INET,SOCK\_STREAM,0))==-1)

bail("socket()");

optlen=sizeof(sr);

int ret=setsockopt(listen\_fd,SOL\_SOCKET,SO\_REUSEADDR,&sr,optlen);

if(ret)

bail("socketopt()");

memset(&srvaddr,0,sizeof srvaddr);

srvaddr.sin\_family=PF\_INET;

srvaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

srvaddr.sin\_port=htons(port);

if((bind(listen\_fd,(struct sockaddr \*)(&srvaddr),sizeof srvaddr))==-1)

bail("bind()");

if(listen(listen\_fd,5)==-1)

bail("listen()");

maxfd=listen\_fd;

if(init(listen\_fd))

bail("init()");

printf("Server is waiting for accertance of new client\n");

for(;;){

rfds=orfds;

n=select(maxfd+1,&rfds,NULL,NULL,NULL);

for(;n>0;n--){

list\_for\_each\_entry(p\_so,&fd\_queue,list){

if(FD\_ISSET(p\_so->fd,&rfds))

p\_so->do\_task(p\_so);

}}}

close(listen\_fd);

return 0;

}

int send\_reply(struct sockfd\_opt \*p\_so){

char reqBuf[BUFSIZE];

char dtfmt[BUFSIZE];

time\_t td;

struct tm tv;

int z;

if((z=read(p\_so->,reqBuf,sizeof(reqBuf)))<=0){

close(p\_so->fd);

FD\_CLR(p\_so->fd,&orfds);

list\_del(&p\_so->list);

free(p\_so);

if(z<0&&errno!=ECONNRESET)

bail("read()");

}else{reqBuf[z]=0;

int num=0;

for(int j=0;j<z;j++){

if(reqBuf[j]>='a'&&reqBuf[j]<='z'||reaBuf[j]>='A'&&reqBuf[j]<='Z')

num++;}

z=write(p\_so->fd,&num,sizeof(int));

if(z<0)

bail("write()");}

return 0;}

int creat\_conn(struct sockfd\_opt \*p\_so){

struct sockaddr\_in cliaddr;

int conn\_fd;

socklen\_t sin\_size;

sin\_size=sizeof(struct sockaddr\_in);

if((conn\_fd=accept(p\_so->fd,(struct sockaddr \*)(&cliaddr),&sin\_size))==-1){

fprintf(stderr,"Accept error:%s\a\n",strerror(errno));

exit(1);}

fprintf(stdout,"Server got connection from%s: %d\n",inet\_ntoa(cliaddr.sin\_addr),ntohs(cliaddr.sin\_port));

if(num++==FD\_SETSIZE){

fprintf(stderr,"too many clients!\n");

close(conn\_fd);

return -1;}

if((p\_so=(P\_SKOPT)malloc(sizeof(SKOPT)))==NULL){

perror("malloc");

return -1;}

p\_so->fd=conn\_fd;

p\_so->do\_task=send\_reply;

list\_add\_tail(&p\_so->list,&fd\_queue);

FD\_SET(conn\_fd,&orfds);

if(conn\_fd>maxfd)

maxfd=conn\_fd;return 0;}

int init(int sk){

P\_SKOPT p\_so;

num=0;

if((p\_so=(P\_SKOPT)malloc(sizeof(SKOPT)))==NULL){

perror("malloc");

return -1;

}

assert(list\_empty(&fd\_queue));

p\_so->do\_task=creat\_conn;

p\_so->fd=sk;

list\_add\_tail(&p\_so->list,&fd\_queue);

num++;

FD\_ZERO(&orfds);

FD\_SET(sk,&orfds);

return 0;}

实验源码三：basic.Client.c

同3.2源码，此处不再赘述。

5、总结与讨论；

在linux操作系统平台下,用C语言进行了I/O多路复用服务器网络编程,熟练掌握了linux下的I/O多路复用服务器的基本编程方法和select函数以及工作模式。