南昌大学实验报告

姓名：衷琪

学号：6103115134

邮箱地址：1483999348@qq.com

专业班级：计算机科学与技术154班

实验日期：2018/5/21

课程名称：Linux程序设计实验

实验项目名称

Socket It Out

实验目的

1、了解套接字的机制
2、尝试学习一些C语言
3、了解网络编程的过程

实验基础

1、需要使用套接字接口来编程通信。
（1）套接字机制：
对于内核来说，套接字是通信的终点。对于应用程序，套接字是允许应用程序使用的文件描述符从/向网络读/写。
（2）bind函数
第一个参数sockfd是在创建socket套接字时返回的文件描述符，第二个参数是struct sockaddr类型的数据结构，由于struct sockaddr数据结构类型不方便设置，所以通常会通过对truct sockaddr_in进行设置，然后进行强制类型转换成struct sockaddr类型的数据。
（3）listen是socket处于监听模式，且同时创建一个socket输入数据队列，将到达的服务请求保存在这个队列中，等待程序处理。
（4）accept,用于server接受客户请求。

实验步骤

1、任务1-套接字：
在一台主机上编写C / S通信，在服务器端回应来自客户端的输入，使用套接字接口来实现这样的过程，使用Linux C进行。
2、任务2：使用Golang重复任务1中内容。

实验数据或结果

1、任务1-套接字： 首先，由于实验中用到了头文件"csapp.h"，"csapp.h"内包含许多头文件，使用起来比较方便，所以先导入"csapp.h"和"csapp.c"。
（1）下载"csapp.h"和"csapp.c",并将它们放在`/usr/include`文件目录下，gcc编译的时候必须 －lpthread,如`gcc -o server.o server.c －lpthread`否则会报错，如下图所示：

```
zhongqi134@ubuntu:~$ cd /usr/include
zhongqi134@ubuntu:/usr/include$ ls
aio.h           error.h         malloc.h        pwd.h           syslog.h
aliases.h       execinfo.h      math.h          python2.7       tar.h
alloca.h        fcntl.h         mcheck.h        rdma            termio.h
argp.h          features.h      memory.h        re_comp.h       termios.h
argz.h          fenv.h          mntent.h        regex.h         tgmath.h
ar.h            fmtmsg.h        monetary.h      regexp.h        thread_db.h
arpa            fnmatch.h       mqueue.h        resolv.h        time.h
asm-generic     fstab.h         mtd             rpc             ttyent.h
assert.h        fts.h           nautilus-sendto rpcsvc          uapi
autosprintf.h   ftw.h           net             sched.h         uchar.h
byteswap.h      _G_config.h     netash          scsi            ucontext.h
c++             gconv.h         netatalk        search.h        ulimit.h
complex.h       getopt.h        netax25         semaphore.h     unistd.h
cpio.h          gettext-po.h    netdb.h         setjmp.h        ustat.h
crypt.h         glob.h          neteconet       sgtty.h         utime.h
csapp.c         gnu-versions.h  netinet         shadow.h        utmp.h
csapp.h         grp.h           netipx          signal.h        utmpx.h
ctype.h         gshadow.h       netiucv         sound           values.h
dbus-1.0        i386-linux-gnu  netpacket       spawn.h         video
dialog.h        iconv.h         netrom          stab.h          wait.h
```

（2）实现流程：在server中调用open*listenfd()函数，建立套接字，绑定端口，并设置为监听模式。同时client调用open*listenfd()发送连接请求；server接收来自client的请求，成功建立连接后。可以通过调用Rio*writen()和Rio*readlineb()进行通信，直到连接关闭。 (3)其中client端截图如下所示，并附上源代码如下：

zhongqi134@ubuntu: ~/exp6

```c
#include <stdio.h>
#include <csapp.h>


#include "csapp.h"
/* usage: ./echoclient host port */
int main(int argc, char **argv)
{
int clientfd, port;
char *host, buf[MAXLINE];
rio_t rio;
host = argv[1];
port = atoi(argv[2]);
clientfd = Open_clientfd(host, port);
Rio_readinitb(&rio, clientfd);
while (Fgets(buf, MAXLINE, stdin) != NULL) {
Rio_writen(clientfd, buf, strlen(buf));
Rio_readlineb(&rio, buf, MAXLINE);
Fputs(buf, stdout);
}
Close(clientfd);
exit(0);
}
"client.c" 23L, 463C                                          7,1
```

```c
        #include <stdio.h>
```

```
#include <csapp.h>
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;
    host = argv[1];
    port = atoi(argv[2]);
    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);
    while (Fgets(buf, MAXLINE, stdin) != NULL) {
    Rio_writen(clientfd, buf, strlen(buf));
    Rio_readlineb(&rio, buf, MAXLINE);
    Fputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

(4)server端截图如下所示，并附上源代码如下：

```
int main(int argc, char **argv) {
        int listenfd, connfd, port, clientlen;
        struct sockaddr_in clientaddr;
        struct hostent *hp;
        char *haddrp;
        port = atoi(argv[1]); /* the server listens on a port passed
        on the command line */
        listenfd = open_listenfd(port);
        while (1) {
                clientlen = sizeof(clientaddr);
                connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
                hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_add
                        sizeof(clientaddr.sin_addr.s_addr), AF_INET);
                haddrp = inet_ntoa(clientaddr.sin_addr);
                printf("server connected to %s (%s)\n", hp->h_name, haddrp)
                echo(connfd);
                Close(connfd);
        }
}
                                                                34,1
```

```
#include "csapp.h"
#include <stddef.h>
#include <stdlib.h>

void echo(int connfd)
{
    size_t n;
    char buf[MAXLINE];
    rio_t rio;
    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
        printf("server received %lu bytes\n", n);
        Rio_writen(connfd, buf, n);
    }
}

int main(int argc, char **argv) {
```

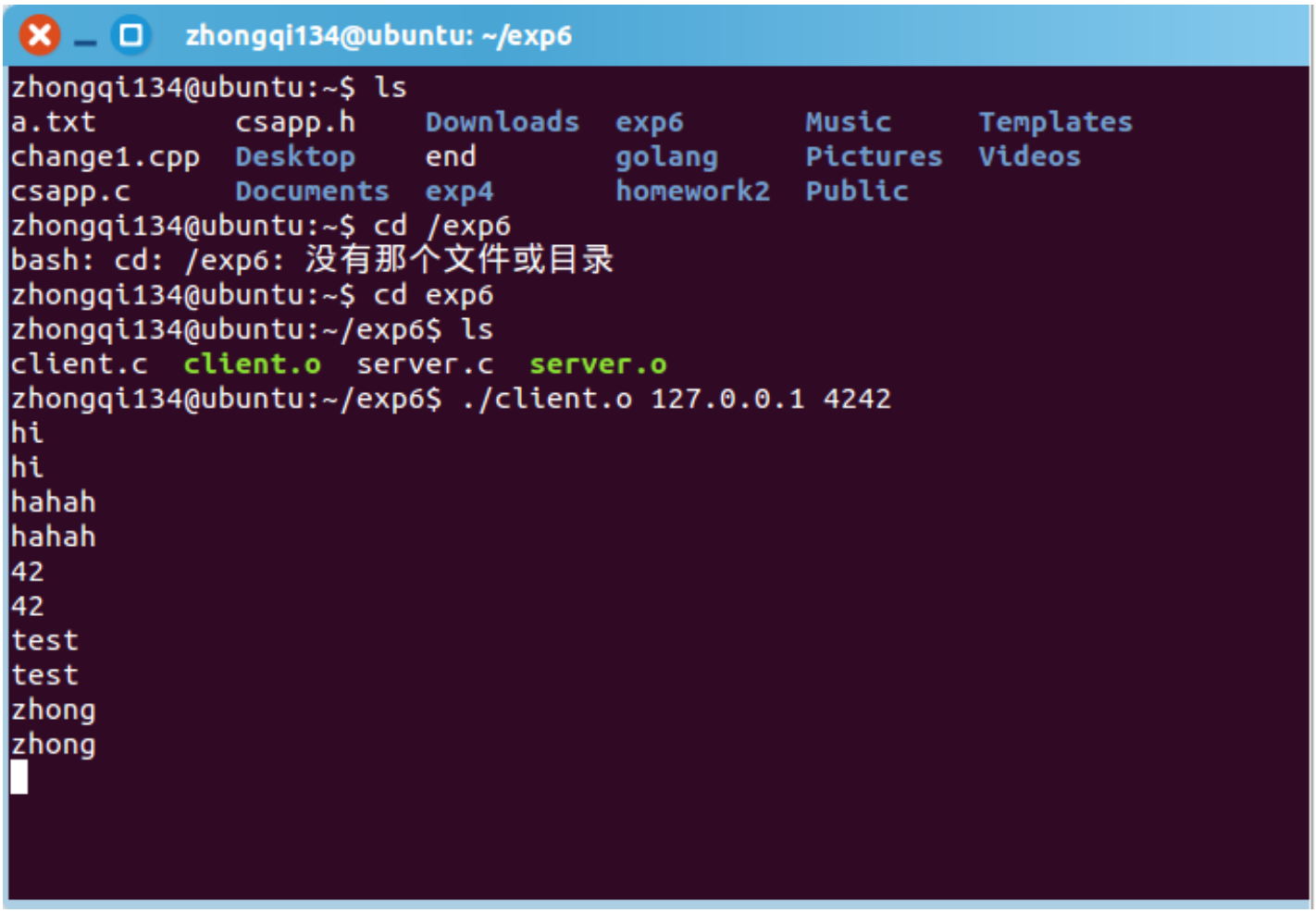
```
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    struct hostent *hp;
    char *haddrp;
    port = atoi(argv[1]);

    listenfd = open_listenfd(port);
    while (1) {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
        hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        haddrp = inet_ntoa(clientaddr.sin_addr);
        printf("server connected to %s (%s)\n", hp->h_name, haddrp);
        echo(connfd);
        Close(connfd);
    }
}
```

（5）分别对"client.c"和"server.c"进行编译，编译结果如下所示：

```
root@ubuntu:/home/zhongqi134/exp6# gcc -o client.o client.c -lpthread
root@ubuntu:/home/zhongqi134/exp6# ls
client.c  client.o  server.c  server.o
```

分别运行"client.o"和"server.o"，

```
root@ubuntu:/home/zhongqi134/exp6# ./server.o 4242
server connected to localhost (127.0.0.1)
server received 3 bytes
server received 6 bytes
server received 3 bytes
server received 5 bytes
server received 6 bytes
```



2、任务2：使用Golang重复任务1中内容。

（1）继实验四配置的golang环境下进行编码。程序功能是可以实现从clinet端输入数据，server能够接收到数据，在将数据返回给clinet接收。 主要实现思路与任务一中大致相同， （2）client中提示输入数据，并显示传回数据。其中client端代码如下：

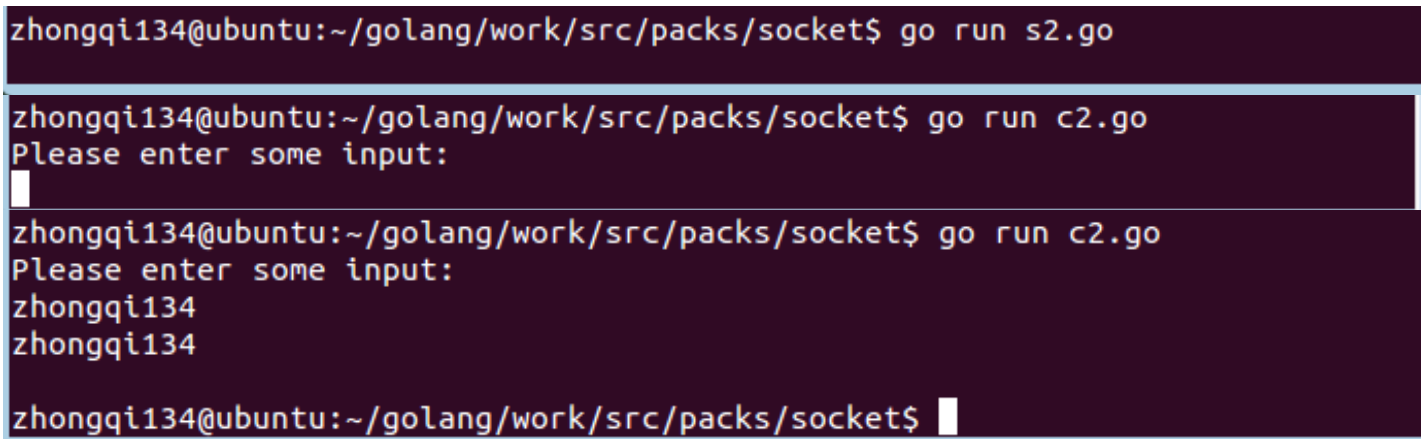
```
package main

import (
    "bufio"
    "fmt"
    "log"
    "net"
    "os"
)

func checkerror(err error){
    if err != nil {
    log.Fatal(err)
    }
}


func main() {
    var (
    //inputReader指向 bufio.Reader 的指针。
    inputReader *bufio.Reader
    input string
    err error
)
inputReader = bufio.NewReader(os.Stdin)
fmt.Printf("Please enter some input: \n")
input, err = inputReader.ReadString('\n')
checkerror(err)
```

```
    conn, err := net.Dial("tcp",":2300")
    checkerror(err)
    defer conn.Close()
    fmt.Fprintf(conn,input)
    res, err := bufio.NewReader(conn).ReadString('\n')
    checkerror(err)
    fmt.Println(string(res))
}
```

（3）server端获得数据，并将数据传回,代码如下：

```
    package main

    import (
        "bufio"
        "fmt"
        "log"
        "net"
    )

    func handle(conn net.Conn) {
        defer conn.Close()
        data, err := bufio.NewReader(conn).ReadString('\n')
        checkerror(err)
        fmt.Println(string(data))
        fmt.Fprintf(conn, data)
        data, err = bufio.NewReader(conn).ReadString('\n')
        checkerror(err)
        fmt.Println(string(data))
    }
    func checkerror(err error){
        if err != nil {
        log.Fatal(err)
        }
    }
    func main() {
        l, err := net.Listen("tcp", ":2300")
        checkerror(err)
        defer l.Close()
        for {
            conn, err := l.Accept()
            checkerror(err)
            go handle(conn)
        }
    }
```

（4）测试代码运行结果，输入数据"zhongqi134"可以看到，返回相同的数据，表明此时能够server和client可以相互通信。运行结果截图如下：



实验思考

1、 通信中可以通过ip地址，传输层协议来区分程序进程间的网络通信和连接，所以在程序编写中应该注意确定它们的值。
2、 这次实验c语言编程主要是参考了老师上课讲的内容，其主要实现的过程如下：首先由server端建立socket，且在client中建立socket，而后建立监听用到的函数为listen，然后通过accpet生成一个新的套接口描述符，来接收客户的连接请求；如果此时client发送请求连接connect，server端接收请求，并且连接成功后，server可向套接字上进行数据发送，client可以接收到server发送的数据，最后关闭server端和client端。

参考资料

1. Linux NetworkProgramming
2. 《深入理解计算机系统》关于csapp.h和csapp.c的编译问题
3. Golang的交互模式进阶
4. go语言socket编程