实验报告1

黄潇颖 2020201622

1. 实验简述

分别实现了C语言和Go语言下的客户端和服务器程序，根据实验报告规范完成了通过Internet发送和接收的测试。

1. 实验细节

Client功能：创建socket并从某一确定端口向server发送从标准输入流中输入的信息。

Server功能：监听某一确定端口并接收来自client的信息，从标准输出重定向到输出文件。

1. C语言下client

首先声明元数据信息和buffer。

int sockfd, numbytes;

char buf[SEND\_BUFFER\_SIZE];

struct *addrinfo* hints, \*servinfo, \*p;

int rv;

char s[INET6\_ADDRSTRLEN];

然后遍历server信息，找到第一个可以连接的server创建socket并尝试连接。连上后释放server信息表，因为后续不会再用上了。

接下来在循环中不断从标准输入中读取SEND\_BUF\_SIZE=2048大小的数据并给server端发送。

while (1)

{

int readinsize = readin(buf);

int bytes\_send = send(sockfd, buf, readinsize, 0);

// sending fail

if (bytes\_send == -1)

{

perror("send");

break;

}

// partial send

// try till success

if (bytes\_send < readinsize)

if (sendall(sockfd, buf, &bytes\_send) == -1)

{

perror("send");

break;

}

if (readinsize == 0)

break;

}

安全性实现：若因为某些原因只有部分数据发送出去了，即partial sending，使用sendall函数来尽力将所有数据都发送完毕。Sendall函数是传入套接字号、buffer、以发送的数据长度的地址和应该发送的数据长度。

int sendall(int *s*, char \**buf*, int \**len*, int *should\_len*)

{

int bytesleft = *should\_len* - \**len*; // how many we have left to send

int n;

while (bytesleft > 0)

{

n = send(*s*, *buf* + \**len*, bytesleft, 0);

if (n == -1)

{

break;

}

\**len* += n;

bytesleft -= n;

}

return n == -1 ? -1 : 0; // return -1 on failure, 0 on success

}

1. C语言下server

Server获取元数据信息部分和client相差不大，获取和封装好socket之后使用socketfd设置端口监听。

listen(sockfd, QUEUE\_LENGTH)

然后进入循环等待连接。若有连接进入，则先创建新的套接字，

new\_fd = accept(sockfd, (struct *sockaddr* \*)&their\_addr, &sin\_size);

然后展开一个RECEIVE\_BUF\_SIZE=2048的buffer，然后进入小循环，不断接收client发来的数据，直到传过来的数据量为0。

while (numbytes != 0)

{

if (numbytes == -1)

{

perror("recv");

numbytes = recv(new\_fd, rev\_buffer, RECV\_BUFFER\_SIZE, 0);

continue;

}

fwrite(rev\_buffer, numbytes, 1, stdout);

fflush(stdout);

numbytes = recv(new\_fd, rev\_buffer, RECV\_BUFFER\_SIZE, 0);

}

1. Go语言下client

Go语言的操作更为简单。首先创建连接：

conn, err := net.Dial(TCP, server\_ip+":"+server\_port)

然后进入循环读取标准输入，并向server发送数据。

readbytes, \_ := os.Stdin.Read(buffer)

sendbytes, err := conn.Write(buffer[:readbytes]) // send

同样对partial sending进行了处理，只需要进行slice操作从buffer得到没发送出去的数据并开循环尽力送出。

1. Go语言下server

首先声明监听变量：

listen, li\_err := net.Listen(TCP, "127.0.0.1:"+server\_port)

然后监听客户端连接。

conn, err := listen.Accept() // 监听客户端的连接请求

然后不断接收client传来的数据直到成功接收的数据量为0.

1. 实验结果







