实验内容

- 实现一个能模拟30%客户端数据包会丢失的服务器代码。对于“收到”的数据包，返回一个成功的消息给客户端。
- 实现一个UDP客户端，向服务器发送10次ping。客户端等待服务器回答的时间至多为1秒，如果一秒内没有收到服务器的回复，则输出“请求超时”，反之则打印出RTT（往返时延）的值。

实验过程

服务器代码`udpserver.c`:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <string.h>
#include <sys/time.h>

/*
@变量声明
*/
int server_sockfd,server_len;
struct sockaddr_in server_address;

/*
@宏定义
*/
#define bufmaxlen 1024 //1KB
#define urllen 50

/*
@brief:生成服务器端的socket描述符
*/
int get_UDPsever_sockfd(void)
{
   return socket(AF_INET,SOCK_DGRAM,0);
}

/*
@brief:设置服务端的sockaddr_in结构，包括协议族，本机地址，端口号，地址长度
*/
void name_server_socker_addr(void)
{
   server_address.sin_family = AF_INET;
   server_address.sin_addr.s_addr = htonl(INADDR_ANY);
   server_address.sin_port = htons(8080);
```

```
    server_len = sizeof(server_address);
}

/*
@brief:信号异常处理
*/
void handle_signal(int sign){
    close(server_sockfd);
    fputs("\nsignal interrupt,safe exit\n",stdout);
    exit(0);
}


int main()
{
    signal(SIGINT,handle_signal);//按下Ctrl+C的时候的处理

    server_sockfd=get_UDPsever_sockfd();
    name_server_socker_addr();

    int bindsta=bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    if (bindsta==-1)
    {
        printf("bind fail\n");
        return -1;
    }

    int  rannum;//生成的随机数
    while (1)
    {
        char buf[bufmaxlen];

        rannum=rand()%10;
        int revflag=recvfrom(server_sockfd,buf,sizeof(buf),0,(struct
sockaddr*)&server_address,&server_len);
        if(revflag==-1)
        {
            printf("UDP 服务器接收错误\n");
            continue;
        }
        if(rannum>2 )
        {
            printf("%s",buf);
            sprintf(buf,"%s","From Server:ping success");
            sendto(server_sockfd,buf,sizeof(buf),0,(struct sockaddr*)&server_address,server_len);
        }
    }

    printf("abnormal exit\n");
    close(server_sockfd);
    return 0;
}
```

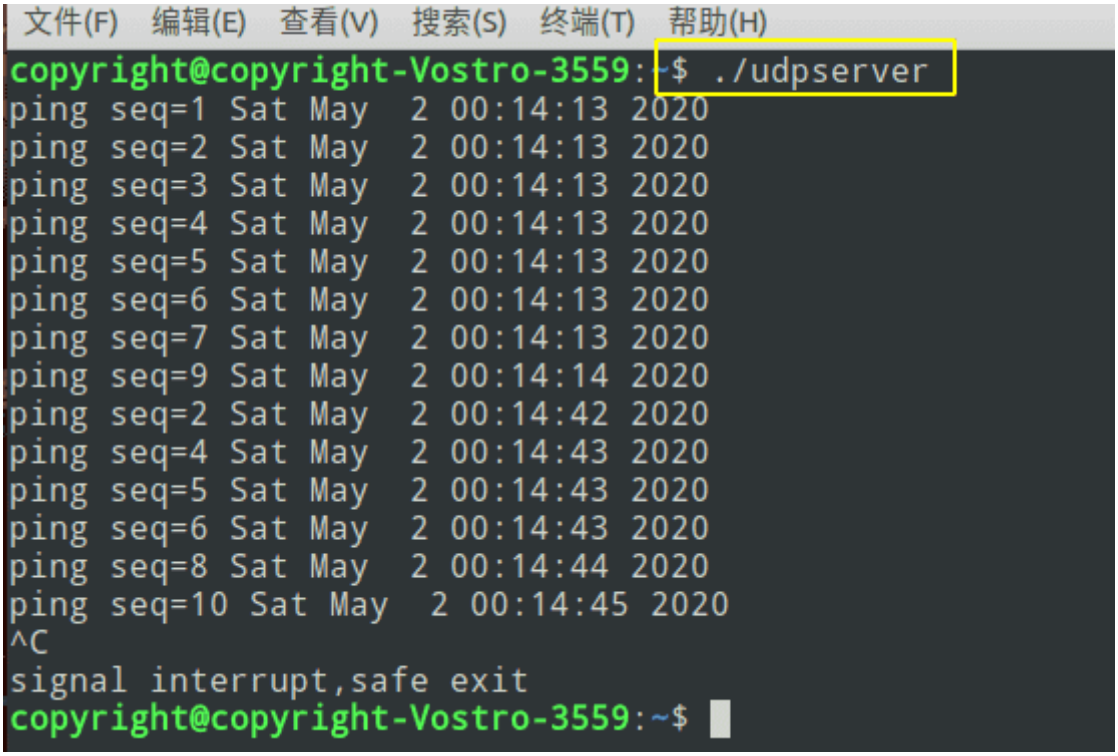
客户端代码 `udpclient.c:`

```
```

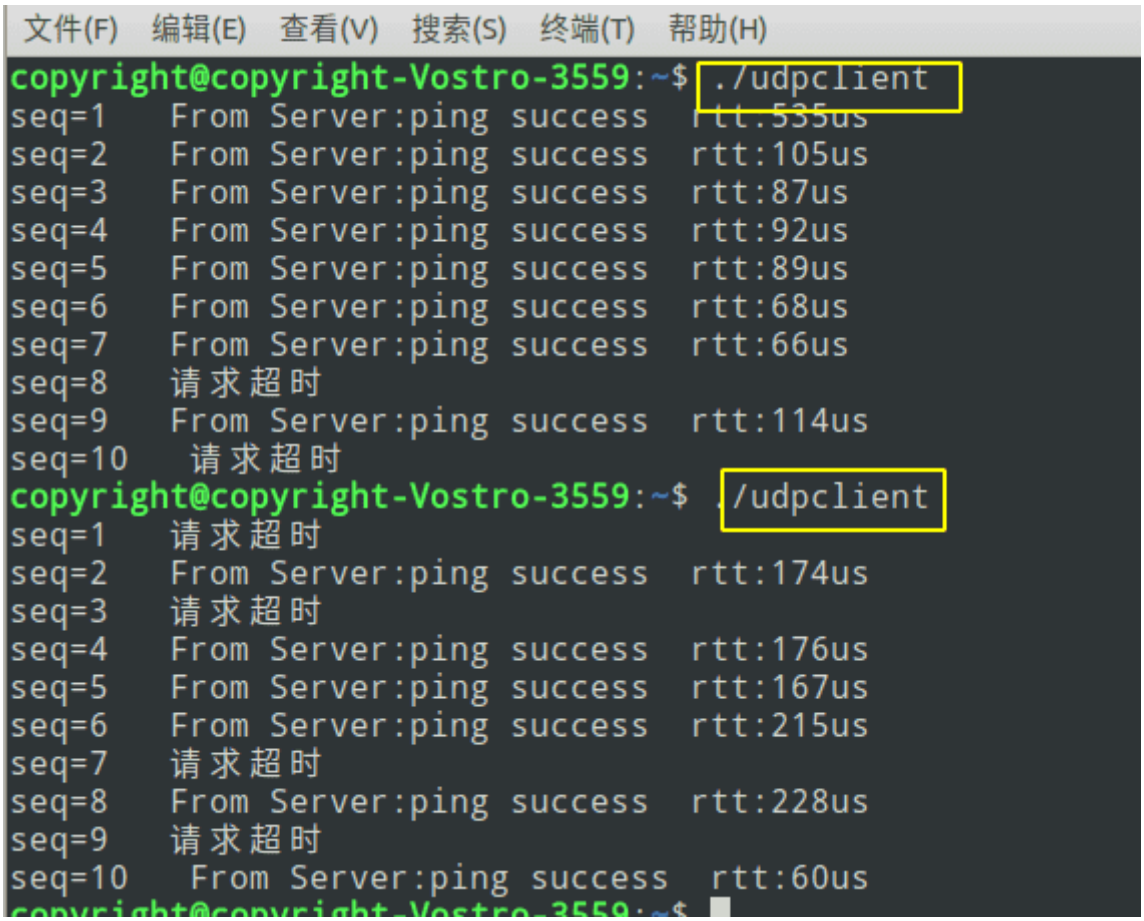
```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <sys/time.h>

#define maxlen 1024

/*
@变量声明
*/
int sockfd,len;
struct sockaddr_in address;

/*
@brief:将ping命令放到buf里面
@para:buf bufsize seqnum
@return:none
*/
void ping_cmd(char *buf,int bufsize,int seqnum)
{
   time_t timeval;
   (void)time(&timeval);
   char seq[5];
   sprintf(seq,"%d%s",seqnum," ");
   sprintf(buf,"%s%s%s","ping seq=",seq,ctime(&timeval));//ctime会自动换行,不用加换行符
}


int main(void)
{
/*配置套接字*/
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = inet_addr("127.0.0.1");//inet函数已经实现了字节转换的功能了
   address.sin_port = htons(8080);
   len = sizeof(address);

/*配置发送跟接收的超时时间，避免阻塞*/
   struct timeval timeout = {1,0}; //该结构体在<sys/time.h>头文件里面
   setsockopt(sockfd,SOL_SOCKET,SO_SNDTIMEO,(char *)&timeout,sizeof(struct timeval));
   setsockopt(sockfd,SOL_SOCKET,SO_RCVTIMEO,(char *)&timeout,sizeof(struct timeval));

   struct timeval tv;//计算RTT用到的结构体
   char buf[maxlen];
   int seq=0;
   while (seq!=10)
   {
     seq++;
     printf("seq=%d   ",seq);

/*发送ping命令，并且获取发送的时间，微秒表示*/
     ping_cmd(buf,sizeof(buf),seq);
```

```
    int sentflag=sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr*)&address,len);
    if(sentflag==-1)
    {
      printf("发送超时\n");
      continue;
    }
    gettimeofday(&tv,NULL);
    unsigned long int pingtime=tv.tv_sec*1000000 + tv.tv_usec;

/*接收服务器命令，若超时则打印出请求超时，反之获取接收的时间，微妙表示*/
    int revflag=recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr*)&address,&len);
    if(revflag==-1)
    {
      printf("请求超时\n");
      continue;
    }
    gettimeofday(&tv,NULL);
    unsigned long  int sucess=tv.tv_sec*1000000 + tv.tv_usec;

/*计算RTT*/
    unsigned long  int RTT=sucess-pingtime;

/*打印结果*/
    printf("%s  rtt:%ldus\n",buf,RTT);
  }

  close(sockfd);
  exit(0);
}
```

服务器截图：



客户端截图：



技巧

- linux下c编程没有itoa函数，但是可以用sprintf函数实现相应的功能
- 用continue，能省一个if

参考

- 链接1
- 链接2
- 链接3

实验体会

- 这次其实本质上实现一个UDP的服务器跟客户端，要注意一些TCP与UDP有对应的API，而且UDP跟TCP的socket编程的流程还是稍微有些不一样的。
- 一开始我是先实现一个能相互通信的服务客户，后来在服务器加上随机数模拟丢包的时候，客户端就因为 `recvfrom` 函数阻塞了，幸亏是有一个超时处理机制。不过这也应该放到平常来想，如果给你一个会阻塞的函数，调用的时候要怎么解决超时的问题？
- 这次还是写的是一个循环的服务器，下次争取写一个加入线程操作，支持并发连接的服务器
- 其实UDP客户端也是可以使用 `connect` 函数的，不过一些发送跟接收函数的选择就会有些不同
  - send函数TCP套接字(客户与服务器)或调用了connect函数的UDP客户端套接字
  - sendto函数用于UDP服务器端套接字与未调用connect函数的UDP客户端套接字

- recv函数从TCP连接的另一端接收数据,或者从调用了connect函数的UDP客户端套接字接收服务器发来的数据
- recvfrom函数用于从UDP服务器端套接字与未调用connect函数的UDP客户端套接字接收对端数据