《计算机网络协议开发》实验报告

第三次实验：文字对战游戏编程实验 (Linux)

组长：郭瑞军

成员:郭瑞军　121220028

李石清　121220047

2012级计算机系

邮箱：cruijunguo@gmail.com

cnnjlsq@163.com

时间：2015.3.19

1、 实验目的

熟悉网络应用层协议设计、实现,掌握服务器套接字编程和 pthread 线程编程技术.

2、 实验内容

本次实验我们首先进行了通信协议的设计,然后分别分工进行了客户端和服务器的实现工作.

2.1 通信协议头部的设计

我们设计了一个通用的协议头部来指示不同的通信类型,然后分别定义了不同的类型进行通信.通信协议我们前期使用文本记录在代码中,下面讲述的具体协议均在工程文件夹下的/common/protocol.txt 中规定.

2.1.1 通信头部的规定

本协议定义客户端服务器之间的通信过程.定义了

注册/登录/登出/显示当前在线用户及状态/发起 or 接受挑战/对战/留言/排行榜查询

共 8 个过程中的通信定义.

协议头部分为 id 和 type 两个字段.

id 由一次通信过程的发起者指定,唯一标识一次通信过程.

同一次通信过程的往返数据使用同一个 id.

type 用于表示通信类型,也同时说明了后续的数据大小 type 共有 13 种.

分别是:SGP/RSP/SIN/RIN/SOT/GET/UPD/CHL/RCH/FGT/ANS/END/MSG/TOP.

协议头结构如下:

```
   4   4
| id | type |
```

2.1.2 具体通信过程的规定

2.1.2.1 注册

注册会话由客户端发起,服务器处理后进行返回结果.共2次通信.

第一次:客户度向服务器发出type为SGP的数据包,并包含使用的用户名及密码.

8 16 16

| header | name | password |

第二次:服务器处理后向客户端返回type为RSP的数据,并返回结果,用一个字节表示注册结果为:

S:成功 F:失败(已注册)

8 1

| header | answer |

2.1.2.2登录

登录会话由客户端发起.服务器处理并响应,共2次通信

第一次:客户端发起登录请求type为SIN,并附上自己的用户名和密码.

8 16 16

| header | name | password |

第二次:服务器校验密码后返回登录结果,type为RIN,使用一个字节表示登录结果:

S:成功 F:密码错误

8 1

| header | answer |

2.1.2.3登出

登出会话由客户端发起,服务器默认不返回结果,单向一次通信

客户端发起 type 为 SOT 类型的数据,并附上自己的用户名.服务器接收并将其下线

8 16

| header | name |

2.1.2.4 显示当期在线用户及状态

服务器主动推送 type 为 UDP 类型的当前在线列表以及状态.用户名 name 及其状态 state 表示,state 有两种,F:空闲;B:对战中;

8 16 1

| header | name | state |

或者在用户登录之后主动发送 type 为 GET 类型的报文来获取所有列表(该报文只有 header 而无后续数据).
服务器返回类型为 GET 的在线列表.
length 表示 name 和 state 组的个数.用 int 表示

8 4 16 1

| header | length / | name | state |/

2.1.5 挑战

挑战会话由一方发起,服务器通知另一方,可以选择接收或拒绝,共涉及两个用户共 4 次通信

第一次:A 向服务器发起挑战, type 为 CHL

第二次:服务器向 B 转发挑战,type 为 CHL

第三次:B 回复结果,type 为 RCH

第四次:服务器向A反馈结果,type为RCH

本部分采用的协议结构如下:

```
            8       16              16        1

    | header | initiator | recipient | state |
```

state: 'A':accept : 'R':refuse

2.1.2.6对战

对战中,对战双方向服务器发送自己的出拳,然后等待结果,结果可能是自己的本次输赢以及血量

变化,或者收到对战结束的最终裁定.

step : Y:剪刀 O:包袱 P:锤子

出拳的数据结构如下,type为FGT:

```
            8       16      1

    | header | name | step |
```

若本次未能结束,则收到一次的裁定,type为ANS:

```
            8       1       4

    | header | ans | blood |
```

若本次结束了游戏,则收到的是最终的裁定,type为END:

```
            8       1

    | header | ans |
```

2.1.2.7 留言

留言由一方向服务器发送,服务器推送给另一方.type 为 MSG, 共 2 个用户,2 次通信

留言的协议格式为:

8	16	16	128
header	sender	reciver	mesg

2.1.2.8 排行榜

排行榜由客户端发起查询,服务器返回结果,共 2 次通信,type 为 TOP

查询过程协议结构为:

8
header

服务器返回的结果为:

8	10*20
header	tops

其中 tops 为 10 组 top10 的用户名及胜利数,结构如下:

16	4
name	victory

2.1.3 协议结构定义

上述各部分的协议在代码中我们采用结构体进行定义,代码如下:

```
#pragma pack(1)
struct Header{
	int id;//the message id
```

```
	char type[4];//the message type
};
struct Account{
	char name[16];//user name
	char passwd[16];//user password
};
struct States{
	char name[16];
	char state;
};
struct Challenge{
	char initiator[16];
	char recipient[16];
	char state;//accept or refuse
};
struct Step{
	char name[16];
	char step;// 'x' as Scissors 'P' as paper 'D' as stone
};
struct Answer{
	char ans;
	int blood;
};

struct Message{
	char sender[16];
	char reciver[16];
	char msg[128];
};
struct Top{
	char name[16];
	int victory;
};
struct SGP{
	struct Header head;
	struct Account account;
};
struct RSP{
	struct Header head;
	char ans;
};
struct SIN{
	struct Header head;
	struct Account account;
```

```
};
struct RIN{
	struct Header head;
	char info;
};
struct SOT{
	struct Header head;
	char name[16];
};

struct CHL{
	struct Header head;
	struct Challenge ch1;
};
struct RCH{
	struct Header head;
	struct Challenge ch1;
};
struct FGT{
	struct Header head;
	struct Step step;
};

struct ANS{
	struct Header head;
	struct Answer ans;
};

struct END{
	struct Header head;
	char end;
};

struct MSG{
	struct Header head;
	struct Message msg;
};

struct TOP{
	struct Header head;
	struct Top top10[10];
};
struct UPD{
	struct Header head;
```

```
        struct States state;
};
#pragma pack()
```

2.2 客户端的实现

客户端采用了两个线程,一个线程负责接收键盘用户输入,并采取响应动作,然后等待另一线程在执行完响应任务后将自己唤醒.另一个线程则负责接收网络上包并今昔解析,如果收到一些动作结束的包并负责唤醒另一个线程.

2.3 服务器的实现

服务器采用多线程编程技术,服务器端采用列表记录所有用户,并为每个用户的连接新建一个线程,每个用户在服务器端使用一个结构表示其信息,所有用户记录在数组 users 中,用户的结构如下:

```
struct InterAccount{

        char name[16];

        char passwd[16];

        char state;

        int socketfd;//if online then has a socketfd

        pthread_t tid;//the thread id

        char mate[16];//挑战 id

        int blood;

        pthread_cond_t cond;

        //接下来是一组信号量,counter 是出拳者数量为 2 时进行判断结果.ans 是判断的结果

        //两个线程中只需要一个进行裁决即可.即维护了一致性也节省了运算

        //这组变量在挑战成功时由`挑战者线程初始化,并由后出拳者进行销毁操作

        pthread_cond_t *cond_fgt;

        pthread_mutex_t *mutex_fgt;

        int *counter_fgt;

        char *goon;
```

```
        char *ans;

        int ivin;

        int vin;

};
```

其中各成员变量含义为:

name:用户名 ;passwd:密码;state:用户状态('F':空闲,'B':对战,'O':离线);socketfd:用户的 socket 描述符;tid:线程 id;mate:对战的对手;blood:剩余血量;cond:挑战用户所用的信号量;cond_fgt:对对战等待出拳使用的信号量;mutex_fgt 出拳用的锁;counter_fgt 出拳者数量计数器指针;goon:继续游戏指示器指针;ans:对战结果指针;ivin:当局已胜次数,vin:已赢局数.

其中使用指针类型的数据是因为两个人对战的时候并不能保证两个人的出拳顺序,所以先出拳者的线程要等待后出拳者,并进行裁决,所以对战双方共享这些变量,故用指针指向.这些在发起挑战的时候被初始化.

2.3.1 重要设计

2.3.1.1 对战裁决

由于对战双方并不能预测出拳顺序,因此,我设计将裁决时机设计在后出拳者出,也就是服务器收到出拳信息时检测当前已经出拳的人数,如果对方还没出拳,则将自己的出拳记录在对战共享区 ans 内,然后自己陷入睡眠也即调用 pthread_cond_wait()函数,然后当自己醒来时直接读取 ans 内的对战结果,而如果收到出拳信息时对方已经出拳,则自己进行结果的判定,并将结果记录在 ans 中后调用 pthread_cond_signal()函数唤醒对方.

2.3.1.2 挑战

挑战我们允许多个线程同时向一个线程发起挑战(这是实际可能发生的),因此,当收到挑战信息时,服务器转发给响应的用户后,让挑战发起者等待在挑战的对象的 cond 变量上.然后被挑战者接受或拒绝其中一个挑战信息,若接受则将 mate 填写为其接受的人,然后唤醒所有等待在自己的 cond 上的线程(pthread_cond_broadcast());这样主动挑战的线程醒来后只要检查对方的 mate 是否是自己即可知道被接收还是拒绝.

2.3.1.3 广播上线/下线消息

我们在登录和登出的函数中调用广播,因此,每当有人上线或者下线都会产生一条广播消息通知所用用户.广播消息采用 UPD 类型.

2.3.1.4 超时唤醒与提醒

我们通过 setsocketopt()函数将服务器套接字设置为非阻塞 recv,并在接收时改写 readn 函数,设置检测如果 15s 未读到想要的数据则返回.下面是 readn 函数

```
int readn( int fd, char *bp, size_t len)

{

        int cnt;

        int rc;

        cnt = len;
```

```
	time_t timebegin, timeouter;
	time(&timebegin);
	printf("read time:%ld\n", timebegin);
	while ( cnt > 0 ){
		rc = recv( fd, bp, cnt, 0);
		if ( rc < 0 ){		/* read error? */
			time(&timeouter);
			if((timeouter - timebegin) > 15){//15s超时
				printf("timeout time:%ld\n", timeouter);
				printf("in readn rc is %d\n", rc);
				fflush(stdout);
				return rc;				/*  return  error
*/
			}
			else continue;

		}
		if ( rc == 0 )				/* EOF? */
			return 0;		/* return short count */
		bp += rc;
		cnt -= rc;
	}
```

```
//      printf("readn:%d",len);

        return len;

}
```

也即我们在进入循环之前先获取标准时间,如果一直没有数据,当醒来检查发现超时,则 readn 函数返回.并且每个用户的服务器在接收到返回之后如果是超时引起的返回,则检查当前用户的状态,如果是空闲状态则什么也不作,反之如果是对战状态或者挑战/被挑战状态,则将其设置其超时状态,并返回给客户端消息告知其超时,消息类型为 END,其中 state 被写为 O/T 分别表示挑战超时或者对战出拳超时.然后唤醒所有等待结果的线程.

2.3.1.4 掉线的处理

如果服务器接收返回值为 1,则说明用户掉线,我们先进行与超时返回相同的操作,也即处理好其他等待回应的线程从而防止死锁或永久睡眠.(对战状态时掉线如同超时处理,实际代码中此处也是一起处理的).然后执行下线操作,也即调用 signout 从而达到通知其他用户和服务器状态一致的维护.最后在处理完所有善后工作后,调用 ptread_exit()结束本线程的服务.

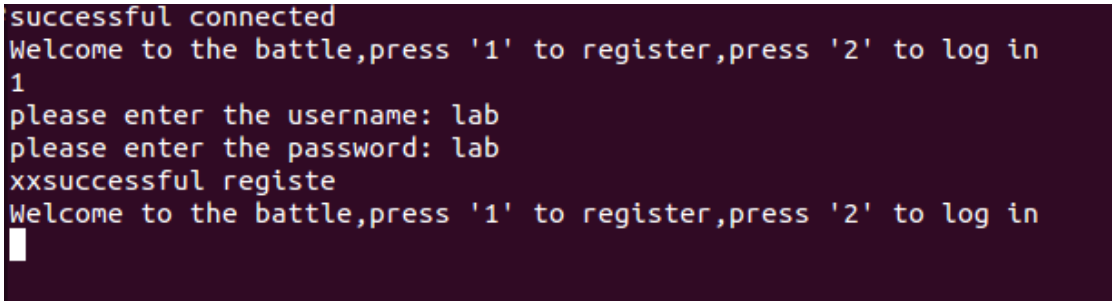
3、 实验结果

我们实现了文字对战游戏要求的所有基本功能并实现了额外的 4 个功能:注册(服务器重启有效),留言,排行榜,登录密码

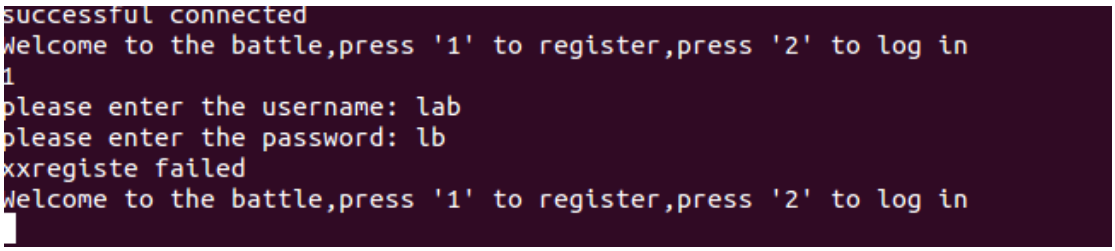
3.1 注册

用户可以使用我们的程序进行选择注册,我们会判断注册成功或失败,并给予显示:

我们注册时要求选择自己用户名和密码.重复用户名会导致注册失败,如下图:注册用户:lab 密码:lab 注册成功.



如果我们重复注册刚才的用户则显示失败:



3.2 登录

我们用我们已经注册的两个用户登录:grj(123), lsq(123)进行登录

```
Welcome to the battle,press '1' to register,press '2' to log in
2
please enter the username: grj
please enter the password: 123
Log in successful
please choose the act you want to behavior:
1.Check the user online and their status
2.Check the top
3.leave message
4.challange others
6.Sign out
```

登录成功后选择功能;

3.3 察看当前在线用户及状态

我们可以察看当前在线用户及其状态,如我们选择 1.

```
1xx4xx3xx1name:lsq   status:Free
name:grj   status:Free
xx2please choose the act you want to behavior:
1.Check the user online and their status
2.Check the top
3.leave message
4.challange others
6.Sign out
please input again
```

我们能够看到当前两个用户在线,状态均为 Free(最前面的 1xx4xx3xx1 是用于调试的信息可以无视之)

3.4 登录/登出广播

```
Welcome to the battle,press '1' to register,press '2' to log in
2
please enter the username: grj
please enter the password: 123
Log in successful
please choose the act you want to behavior:
1.Check the user online and their status
2.Check the top
3.leave message
4.challange others
6.Sign out
name:lsq states:F
```

在第二个用户登录的时候,前面登录的所有用户都可以看到一个即时的提示.

3.5 挑战

我们可以选择挑战一个在线 Free 状态的玩家,玩家可以接收或者拒绝,接收双方进入对战状态,长时间未选择则挑战方认为挑战被拒绝,未选择方做出选择后会提示挑战过期无法接收.

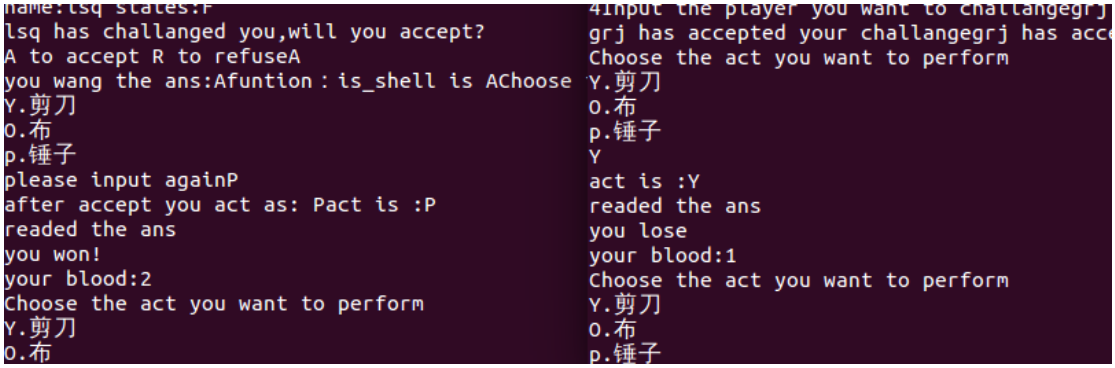
```
4                                                   6.Sign out
4Input the player you want to challangelsq          please input againgrj has challanged you,will you accept?
lsq has accepted your challangelsq has accepted your challange,A to accept R to refuseA
Choose the act you want to perform                  you wang the ans:Afuntion : is_shell is AChoose the act you want to perfo
Y.剪刀                                              Y.剪刀
O.布                                                O.布
p.锤子                                              p.锤子
                                                    please input again
```

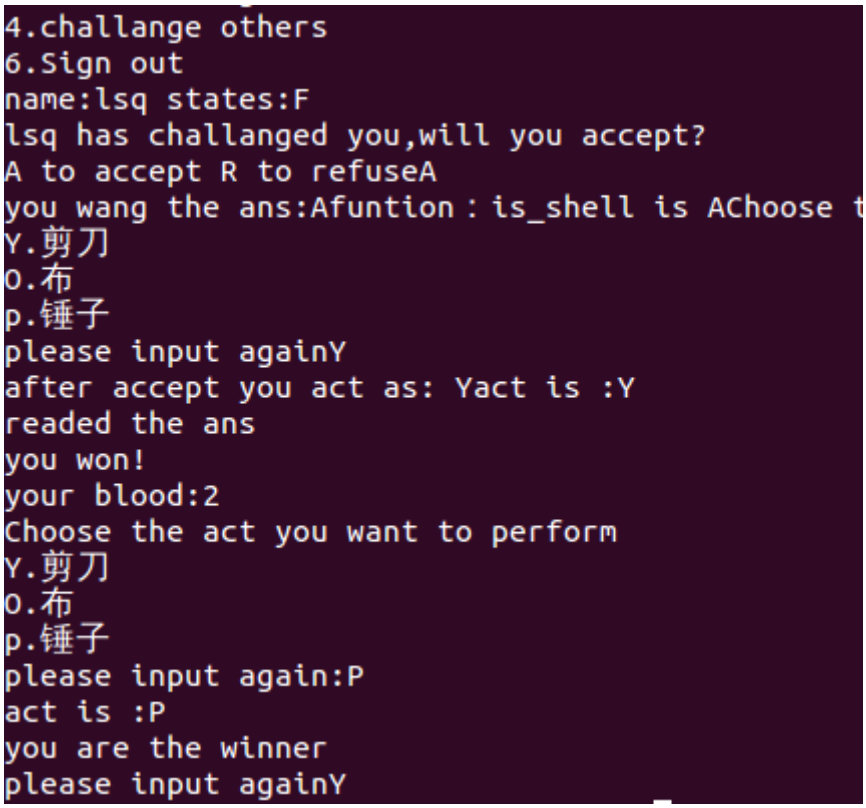
如图为接收挑战进入对战.

3.6 对战

对战采取三局两胜制，当有一方胜 2 局游戏作为一次对战结算.结果刷新服务器的排行榜.

每局有胜负平三种结果.负者扣一点血，胜者不加血，血量扣完一次对战结算.

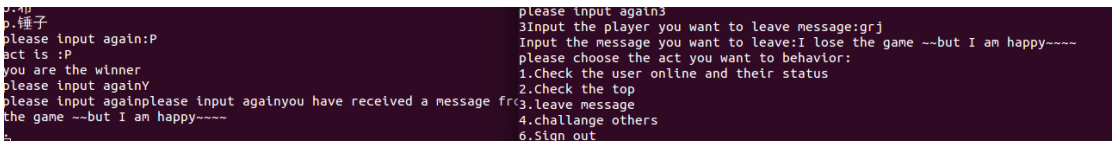




如图，经过 3 局其中一平两胜，游戏结束.

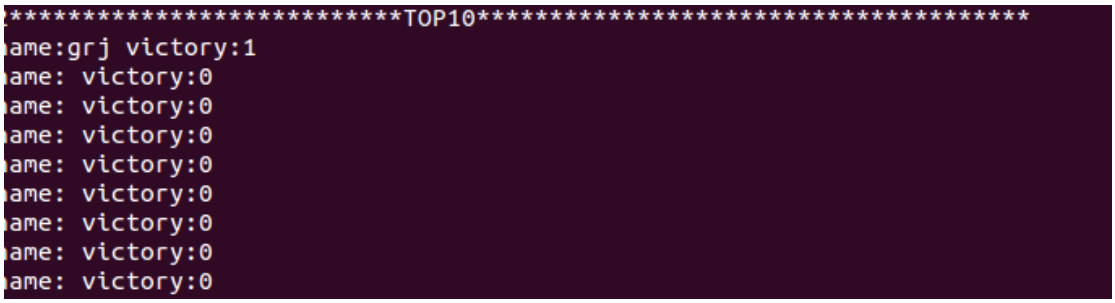
附加功能：

3.7 留言



我们可以选择给在线玩家留言，对方会受到并提醒.

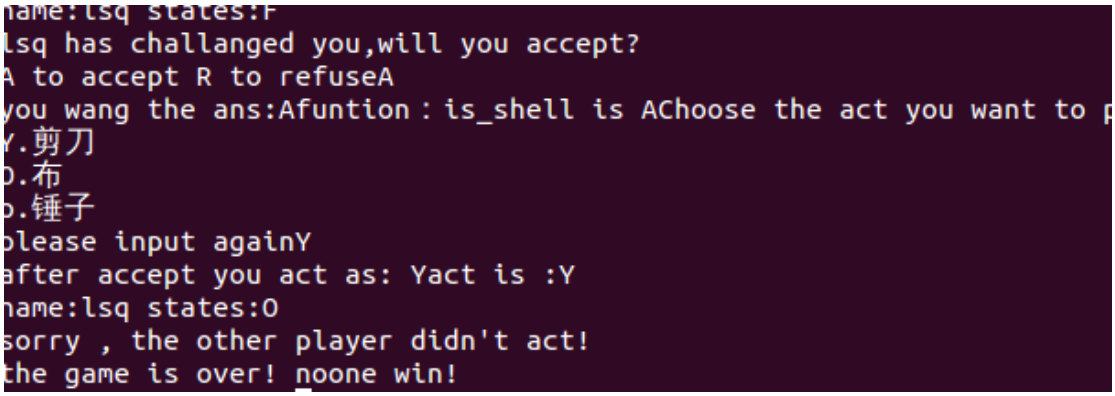
3.8 排行榜



我们也可以查看在线 top10 排行榜,如图经过一次对战,用户 grj 赢得次,,位居榜首,其他并没有玩家赢得游戏.

3.9 掉线和超时

当空闲的一方掉线时会给所有空闲状态的玩家发送广播,因为可以从 recv 返回 0 判断.当超时时提醒玩家对面超时.



4、 实验中遇到的问题及解决方案

4.1.死锁

我们的客户陷入编程中多次遇到死锁,通过不断进行 printf 的方法,我们定位死锁位置,判断除线程睡死的位置,从而定位程序的 bug,从新考虑上锁逻辑.最终得到解决,多线程中 wait 和 signal,lock,unlock 的使用要仔细斟酌其中的各种情况.一定要防止占有锁的线程因其他资源的等待而导致锁永久占有,互相等待.

4.2.多线程读取输入流分配

我们在客户端编程中,使用了两个线程,其中有一个地方会出现两个线程同时请求键盘输入的问题,从而导致输入的混乱.

后来我们发现其中两个线程对输入的请求是有逻辑上的有限级的,因此,我们使用标志位来表征输入的所有权,如果输入被有限级低的线程获取则主动将结果送给另一线程,代码如下:

```
while(1)
        {
            if(want != 'w'){
                pthread_mutex_lock(&mutex_want);
```

```
                        setbuf(stdin, NULL);

                        scanf("%c",&mode2);

                        if(want == 'w'){

                              printf("you wang the ans:%c", mode2);

                              is_shell = mode2;

                              pthread_mutex_unlock(&mutex_want);

                        }

                        else if(mode2 == '1'||mode2 == '2'||mode2 ==
'3'||mode2 == '4'||mode2 == '6'){

                              pthread_mutex_unlock(&mutex_want);

                              break;

                        }

                        else {

                              printf("please input again");

                              pthread_mutex_unlock(&mutex_want);

                        }

                  }

            }
```

其中 want 为优先级输入抢占标志,如果抢占输入,则主动将结果送给 is_shell 中,并且不会再次抢占输入锁,一直到另一方放弃输入权,也即将 want 置位,则该线程继续抢占输入锁.

4.3 超时等待/断线

我们通过设置非阻塞的 recv 然后通过返回值和 errno 进行判断 recv 返回的原因,若果是 0,是断线,负数时 errno 可以判断是否是唤醒的.并通过标准时间的差进行判断时间,超过额定值就发送终止信号给客户端.

5、 实验的启示/意见和建议

1.本次实验深入了解了 socket 的 send 和 recv 的使用,对其属性的设置有了基本的了解.

2.熟悉了 pthread 编程,熟悉使用 pthread_mutex_t,pthread_cond_t 两种 pthread 变

量类型,重新复习了线程间通信的方式,以及死锁防止,临界区保护,资源控制等线程编程技术.

附：本次实验你总共用了多长时间？包括学习时间、编写代码时间和测试时间。（仅做统计用，时间长短不影响本次实验的成绩。）
共用 2 人两周时间,前后包括查阅资料以及 debug,其中 debug 有两天连续到深夜.虽然非常累,但是最终看到程序正常运行,而且自己学到了很多知识,就感觉这些都是值得的,也会感觉很开心.