

# 《计算机网络协议开发》实验报告

## 第四次实验：文字对战游戏编程实验

姓名：吕玉龙

学号：191220076

2019级计算机科学与技术院/系

邮箱：1931015836@qq.com

时间：2022.4.17

# 一、实验目的

熟悉网络应用层协议的设计、实现，掌握服务器套接字编程。

# 二、成品功能

完成了 pdf 中所要求的所有基本功能，当然本服务器只支持最多 15 人同时使用，这个是在代码中可变的。同时提供了两个扩展功能，扩展功能一：用户之间可以互相发送消息，这个消息的接收方可以处于对战中或处于请求对战中；扩展功能二：提供排行榜，用户在客户端直接请求，服务器即会发会排行榜信息。本次实验基于自行实现的 SSP 协议。

# 三、实验内容

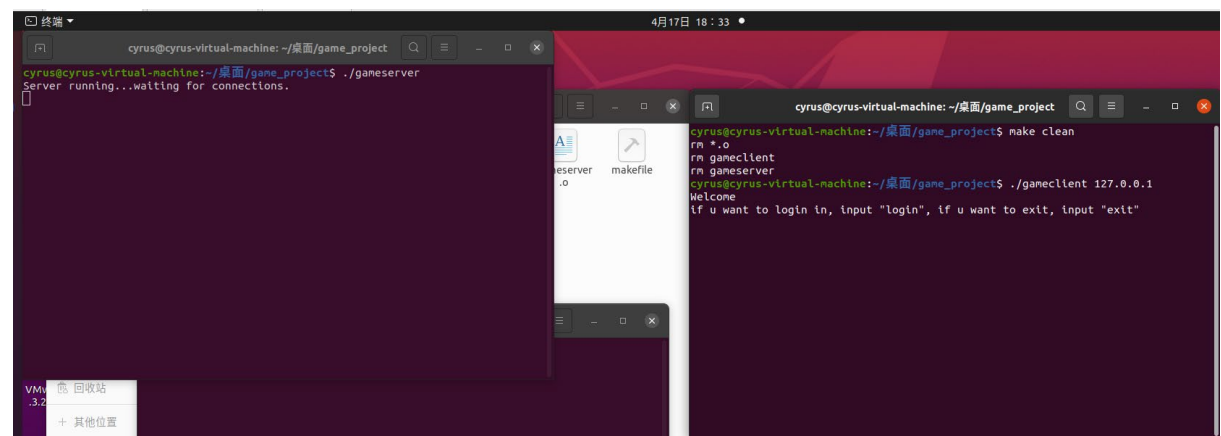
实现思路，参考了 week7 pdf 中给出的实现方式。

## 1) 客户端

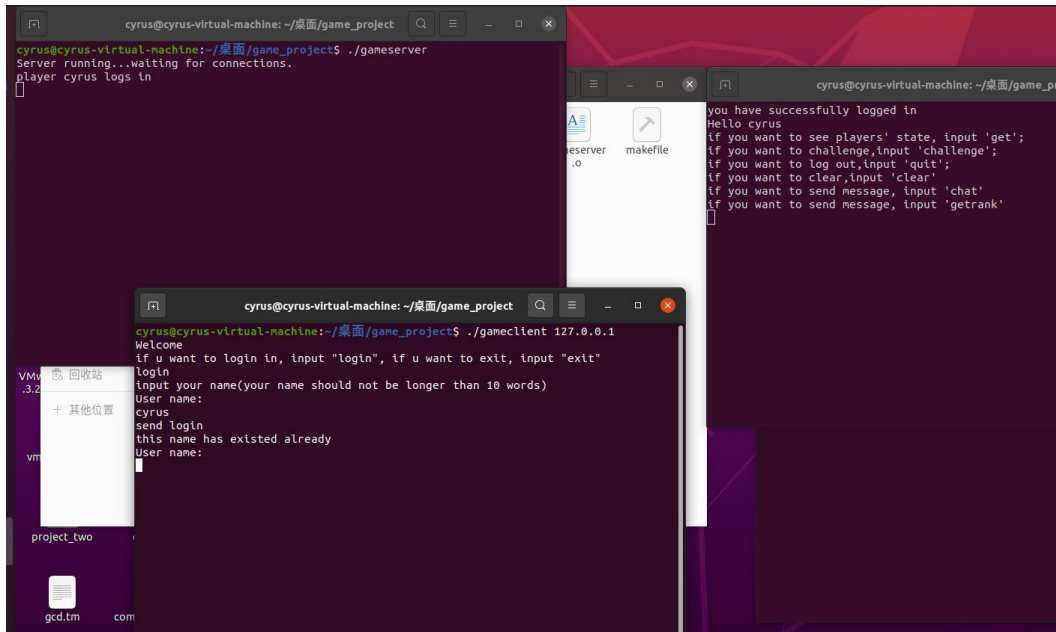
客户端分成了线程，监听线程以及本身的发送线程。两个线程的交互通过共享变量进行交互，同时还有一些互斥锁，其中有一个很重要的互斥锁 `mutex_listen_read`，这个锁可以实现用户输入之后对用户发送的线程的阻塞。例如，当用户输入用户名登录时，会由服务器判断是否存在输入的用户名，这时需要阻塞发送线程，直到服务器发包回来告诉客户端是否合法。

发送线程：

首先是一个登录界面：



输入 login 后，输入用户名，不合法服务器会告诉你不合法，直至输入合法名字，如果输入合法，就进入下图左上角的界面，同时 server 会显示用户登入。代码这里实际上是从 PageLogin 函数跳转到 PageBattle，

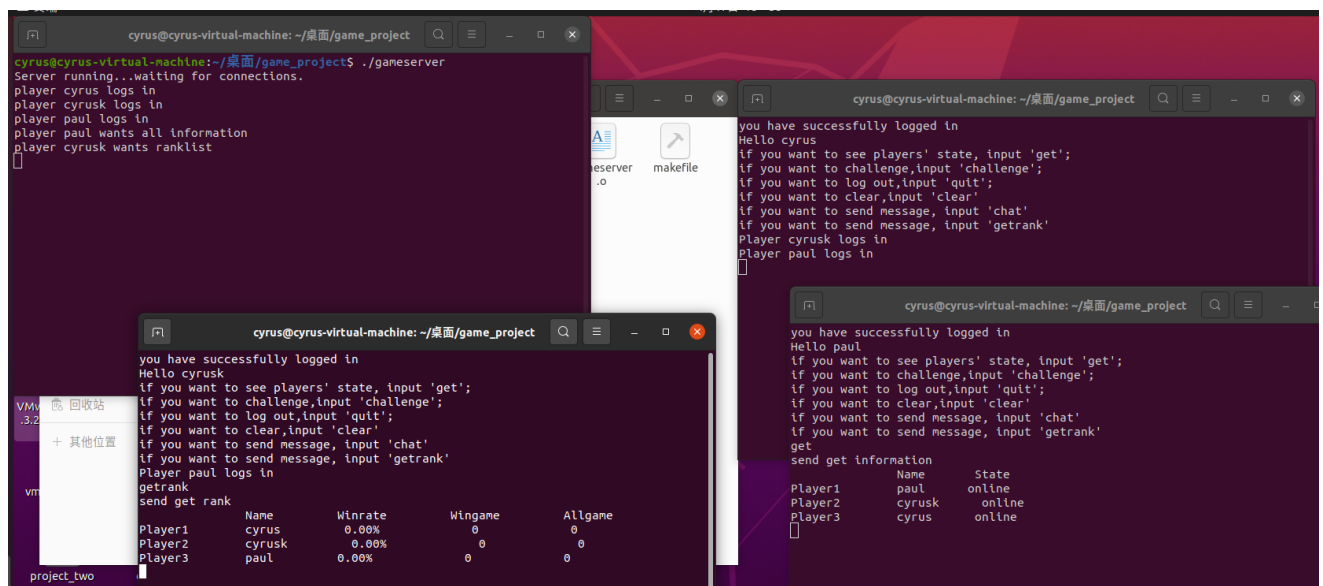


```

528     char sendline[MAXLINE];
529     printf("Welcome\n");
530     while(true){
531         char inputstr[MAXLINE];
532         printf("if u want to login in, input \"login\", if u want to exit, input \"exit\"\n");
533         gets(inputstr);
534         if(strcmp(inputstr,"login") == 0){
535             PageLogin(sockfd);
536             //system("clear");
537             PageBattle(sockfd);
538             system("clear");
539         }
540         else if(strcmp(inputstr,"exit") == 0){
541             printf("thank you for using my\n");

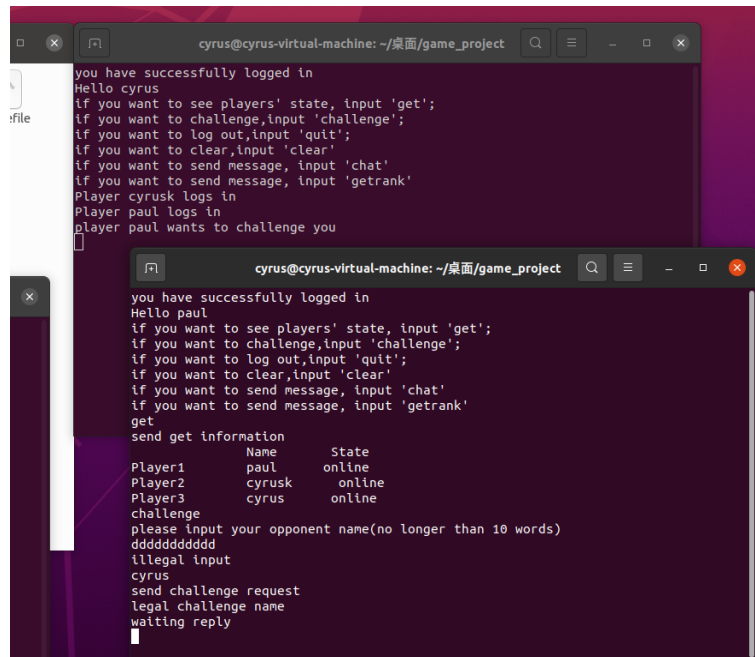
```

这里演示时有三个用户登录，接下来会判断用户的输入，向服务器发包，同时由于监听线程一直运作，所以听到回复时，会解析包，并打印出信息。



这里分别输入了 `get` 和 `getrank`，解析完，客户端效果如上，同时，可以看到，当有其它用户登入之后，客户端也会显示登入信息 `player xxx logs in`，当然这个信息在对战时候同样会显示，因为这个打印是实现在监听线程中。

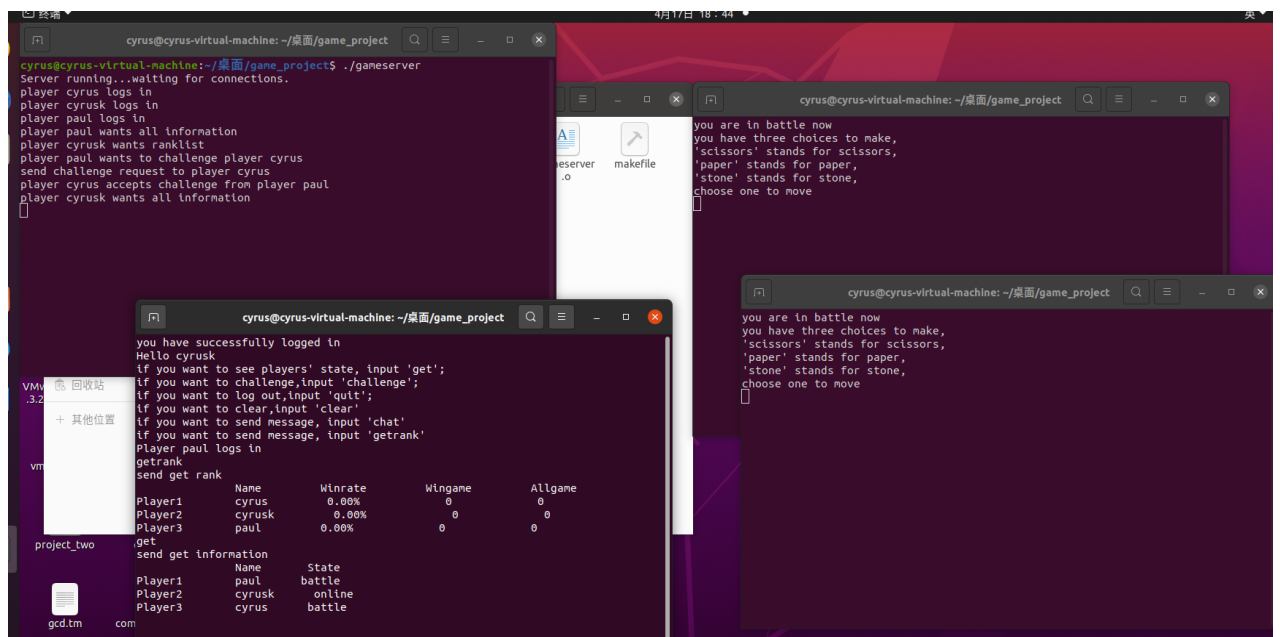
接下来用户想对战的话，输入 **challenge**，接下来需要输入挑战的名字，这里与对用户名的判断一样，由服务器判断，如果合法会 **waiting reply**，对方接收后，由 **PageBattle** 函数跳转到 **PageChall** 进行对战，这几个函数的实现大都类似，这里不予展示



The image shows two terminal windows from a virtual machine. The top window shows the server's initial state and a player named 'paul' logging in and challenging 'cyrus'. The bottom window shows the server's response to the challenge, including a list of players and their states, and a prompt for the opponent's name.

```
you have successfully logged in
Hello cyrus
if you want to see players' state, input 'get';
if you want to challenge, input 'challenge';
if you want to log out, input 'quit';
if you want to clear, input 'clear';
if you want to send message, input 'chat'
if you want to send message, input 'getrank'
Player cyrusk logs in
Player paul logs in
player paul wants to challenge you

you have successfully logged in
Hello paul
if you want to see players' state, input 'get';
if you want to challenge, input 'challenge';
if you want to log out, input 'quit';
if you want to clear, input 'clear';
if you want to send message, input 'chat'
if you want to send message, input 'getrank'
get
send get information
Name      State
Player1   paul    online
Player2   cyrusk  online
Player3   cyrus   online
challenge
please input your opponent name(no longer than 10 words)
dddddddddd
illegal input
cyrus
send challenge request
legal challenge name
waiting reply
```



The image shows three terminal windows. The top window shows the server's initial state and a player named 'paul' logging in and challenging 'cyrus'. The middle window shows the server's response to the challenge, including a list of players and their states, and a prompt for the opponent's name. The bottom window shows the server's response to the challenge, including a list of players and their states, and a prompt for the opponent's name.

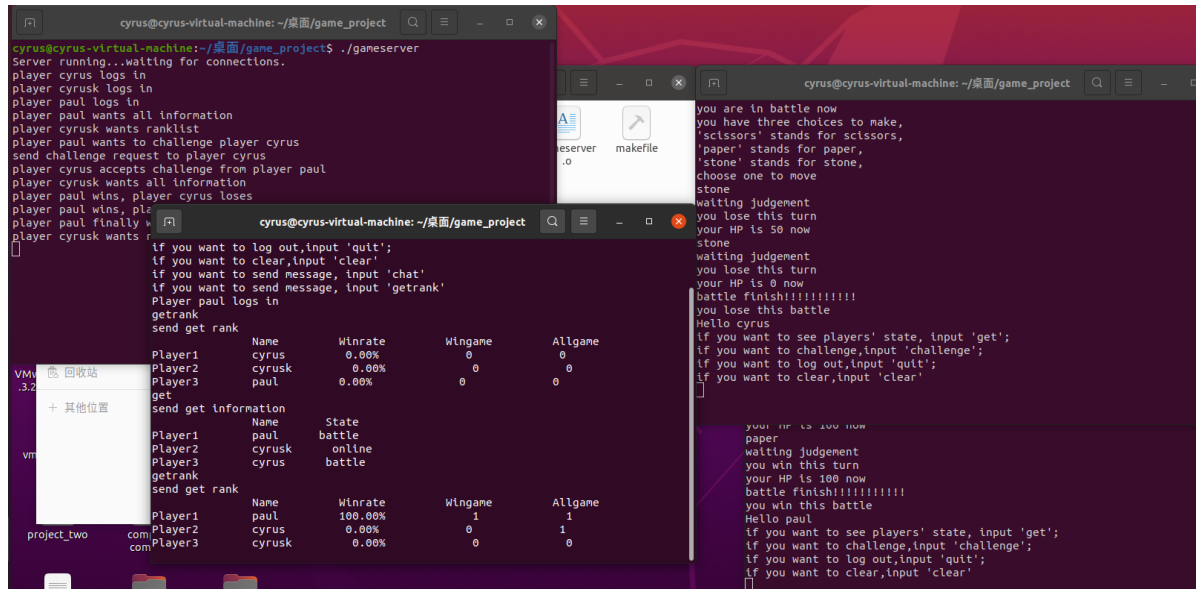
```
cyrus@cyrus-virtual-machine: ~/桌面/game_project
cyrus@cyrus-virtual-machine:~/桌面/game_project$ ./gameserver
Server running...waiting for connections.
player cyrus logs in
player cyrusk logs in
player paul logs in
player paul wants all information
player cyrusk wants ranklist
player paul wants to challenge player cyrus
send challenge request to player cyrus
player cyrus accepts challenge from player paul
player cyrusk wants all information

you are in battle now
you have three choices to make,
'scissors' stands for scissors,
'paper' stands for paper,
'stone' stands for stone,
choose one to move

you are in battle now
you have three choices to make,
'scissors' stands for scissors,
'paper' stands for paper,
'stone' stands for stone,
choose one to move

you have successfully logged in
Hello cyrusk
if you want to see players' state, input 'get';
if you want to challenge, input 'challenge';
if you want to log out, input 'quit';
if you want to clear, input 'clear';
if you want to send message, input 'chat'
if you want to send message, input 'getrank'
Player paul logs in
getrank
send get rank
Name      Winrate  Wingame  Allgame
Player1   cyrus    0.00%    0        0
Player2   cyrusk   0.00%    0        0
Player3   paul     0.00%    0        0
get
send get information
Name      State
Player1   paul    battle
Player2   cyrusk  online
Player3   cyrus   battle
```

另一个客户端这里可以看到两者的状态都变成 **battle**，对战完成之后，会由 **PageChall** 跳回到 **PageBattle** 函数。下面的截图中，客户端会仍存有对战过程，服务器也有所需信息，对战完成后，使用得到排行榜信息，可以看到排行榜也变了



其它功能这里就不予展示，接下来说一下服务器的实现

## 2) 服务器

分成了四个线程

主线程：监听客户连接请求，创建客户线程处理服务器与客户端的通信。

客户线程：在主循环中接收来自客户端的报文，根据不同类型的报文做相应的处理

对战线程

广播线程

客户线程：实际上就是监听线程，与客户端的监听线程实现相似，唯一不同的是转发时候，是在收到的线程中直接进行转发。因为我有一个结构体，存储所有人的信息，可以直接在接受到的线程中进行转发。同时，对于出招报文，登录注册报文会对一些全局变量进行修改。

```
typedef struct Map_Client* Map_Client;
typedef struct Battle* Battle;
typedef struct Ranklist* Ranklist;
enum Client_state{online,request,battle};

struct Map_Client {
    pthread_t tid;
    char username[12];
    enum Client_state state;
    int sockfd;
    Map_Client next;
};
Map_Client Map = NULL;
char updatename[12];
bool isrefresh = false;
bool log;
pthread_mutex_t mutex_map;

struct Battle {
```

这个结构使用了链表形式，存储了用户名对应的描述符，用户状态等等信息。广播线程也使用这个链表。

广播线程：

使用了一个全局变量，isrefresh 来记录全局结构，map 是否有更新。如果有，那么立即进行一个广播。这里统一使用互斥锁 mutex\_map 防止进行死锁。全局变量 updatename 用来记录所需广播的用户名。

对战线程：

这里使用了另一个结构体：

```
35  bool log;
36  pthread_mutex_t mutex_map;
37
38  struct Battle_{
39      bool player1ready;
40      bool player2ready;
41      char player1[12];
42      int sockfd1;
43      int Hp1;
44      int move1;
45      char player2[12];
46      int sockfd2;
47      int Hp2;
48      int move2;
49      bool finish;
50      Battle next;
51  };
52  Battle allbattle = NULL;
53  pthread_mutex_t mutex_battle;
54
55  struct User_State{
```

它记录了对战双方的信息，包括是否出招，血量剩余等等，同样以链表的形式存储。当两个客户端同意对战时，就会添加这一场对战的信息。对战线程对这里链表中的信息进行操作。对战结束之后，这个链表则会删除这场对战的信息。同时对战线程还负责向对战双方发送每次对战出招后的结果以及整场对战的结果。同时对战线程还需负责对排行榜的更新，

```
59
60  struct Ranklist_{
61      int init;
62      char name[12];
63      double winrate;
64      int wingame;
65      int allgame;
66  };
67  struct Ranklist_ ranklist[15];
68  pthread_mutex_t mutex_rank;
69
```

由于排行榜涉及到排序，故使用了结构数组的形式，每场对战结束后，都要更新排行榜，排序使用了 c 库函数 qsort，重载了一个 cmp 算法即可。由于代码比较多，但很多收发的思路都是一样的，故这里只介绍思路

## 四、问题及解决方法

最大的问题在于多线程编程，以前多线程编程的经验很少，所以很多东西都是现学的。同时编写代码过程中，因为对互斥锁的使用不当，出现了死锁的情况。

另一个问题在于如何阻塞线程，因为有时客户端的发送需要让服务器判断，所以就要阻塞发送线程，但是一开始并没有想到这个问题，客户端会直接继续运行，然后被阻塞在 `gets` 上。

另一个设计问题是用户本身的输入，以及对挑战的回应，本来是用两个 `buffer` 来对不同的情况进行读入输出，但是由于设计的不好，它很容易就会读到另一个 `buffer` 中，故最终全部归到 `PageBattle` 的输入 `buffer` 中，增加一个全局变量来判断现在是否正在被挑战，来区分用户的输入应该是对挑战的回应还是自己的输入（例如得到所有玩家的信息等）

比较大的设计问题暂时就这么多，其它一些遇到的小 `bug` 就不放在这里了，写的时候遇到的小 `bug` 也有些，但是很快就解决了，上面都是比较大的设计问题。