typedef struct

{

char username[20];//用户名

int w\_time; //获胜用时

int won\_NUM; //获胜次数

int played\_NUM; //游戏次数

bool is\_record; //是否已被记录

}Leaderboard; //排行榜结构体

typedef struct

{

int adjacent\_mines;//周围地雷数量

bool revealed; //是否被揭示

bool is\_mine; //是否地雷

bool is\_place; //是否标记

}Tile; //瓦片结构体

typedef struct

{

Tile tiles[MUM\_TILES\_X+2][MUM\_TILES\_Y+2];

}GameState; //游戏状态

处理临界问题主要使用以下两个互斥锁

pthread\_mutex\_t mutex1;

pthread\_mutex\_t mutex2;

线程池原理：在应用程序启动之后，就马上创建一定数量的线程，放入空闲的队列中。这些线程都是处于阻塞状态，这些线程只占一点内存，不占用CPU。当任务到来后，线程池将选择一个空闲的线程，将任务传入此线程中运行。当所有的线程都处在处理任务的时候，线程池将自动创建一定的数量的新线程，用于处理更多的任务。执行任务完成之后线程并不退出，而是继续在线程池中等待下一次任务。当大部分线程处于阻塞状态时，线程池将自动销毁一部分的线程，回收系统资源。

服务器程序启动之前，初始化线程池，启动线程池中的线程，由于还没有任务到来，线程池中的所有线程都处在阻塞状态，当一有任务到达就从线程池中取出一个空闲线程处理，如果所有的线程都处于工作状态，就添加到队列，进行排队。如果队列中的任务个数大于队列的所能容纳的最大数量，那就不能添加任务到队列中，只能等待队列不满才能添加任务到队列中。

线程池中使用的结构体

struct job

{

void\* (\*callback\_function)(void \*arg); //线程回调函数

void \*arg; //回调函数参数

struct job \*next;

};

struct threadpool

{

int thread\_num; //线程池中开启线程的个数

int queue\_max\_num; //队列中最大job的个数

struct job \*head; //指向job的头指针

struct job \*tail; //指向job的尾指针

pthread\_t \*pthreads; //线程池中所有线程的pthread\_t

pthread\_mutex\_t mutex; //互斥信号量

pthread\_cond\_t queue\_empty; //队列为空的条件变量

pthread\_cond\_t queue\_not\_empty; //队列不为空的条件变量

pthread\_cond\_t queue\_not\_full; //队列不为满的条件变量

int queue\_cur\_num; //队列当前的job个数

int queue\_close; //队列是否已经关闭

int pool\_close; //线程池是否已经关闭

};

线程池的实现通过以下几个函数：

//初始化线程池

struct threadpool\* threadpool\_init(int thread\_num, int queue\_max\_num);

//向线程池中添加任务

int threadpool\_add\_job(struct threadpool \*pool, void\* (\*callback\_function)(void \*arg), void \*arg);

//销毁线程池

int threadpool\_destroy(struct threadpool \*pool);

//线程池中线程函数

void\* threadpool\_function(void\* arg);

如何编译和运行程序：

编译：

在server和client文件夹下执行make命令编译程序。执行make clean删除程序

运行：

进入server文件夹，执行命令 ./server 12345 运行服务器

进入client文件夹，执行命令 ./client 127.0.0.1 12345 运行客户端（127.0.0.1为本地服务器IP，可根据真实服务器IP自行更改，如：服务器IP为192.168.1.11，.则执行/client 192.168.1.11 12345运行程序）