课程编号：C**********

《Linux 操作系统》
大实验说明文档

姓　　名	丁**	学　　号	********
班　　级	软件 16**	指　导　教　师	**
实　验　名　称	《Linux 操作系统大作业》		
开　设　学　期	2018-2019 第一学期		
开　设　时　间	第 1 周——第 6 周		
报　告　日　期	2018 年 10 月 13 日		
评　定　成　绩		评　定　人	**
		评　定　日　期	****年**月**日

东北大学软件学院

大实验说明

一. 实验题目

网络游戏:游戏的内容不限,量力而行。但绝对不允许抄袭。多个客户可通过 socket 连接到服务器。游戏的业务部分（数据、逻辑）都在服务器端处理。而客户端主要用于接收用户的输入及显示。服务器端要求使用在课程中学到的知识进行编写：用 C/C++编写。加分点比如：进程/线程间通信、异步 I/O、进程/线程同步、实时性游戏所必须的良好的应用层协议等。对客户端编程的语言虽然并不强求，但希望能充分利用上课所学知识（建议使用课上学的 curses 库以及 itimer 和 signal）。

二. 系统设计

本次实验我结合了 epoll 机制，反应堆模型，回调函数机制对服务器端进行编写，并将信号机制，父子进程通信，curses 库，itimer，signal 机制运用到了客户端的程序编写中。

三. 实验功能与创新点

创新点：

（1） 服务器使用 epoll 机制实现了异步 I/O 机制，支持高并发。

（2） 服务器参考开源网络库 libevent 的设计方式，通过回调函数机制保证消息处理的效率。

（3） 客户端通过父子进程通信保证一个进程处理逻辑，一个监听服务器的传输消息，当有消息到来时，使用信号机制通知子进程进行处理。实现类似异步的方式处理消息。

（4） 服务器支持大量用户同时在该服务器上对战，服务器会为每两个客户开辟一个房间进行对战，房间内对战双方数据同步，不同房间数据互不影响。

（5） 服务器随机生成地图，将该地图发送给客户端，保证不同客户端获取不同地图进行对战，提高游戏仿真性。

（6） 客户端采用了多线程处理子弹逻辑，引入了线程的同步机制，保证共享数据的一致性

（7） 除子弹移动逻辑外，坦克的移动，开火，碰撞检测，回血，子弹发射，得分，失血，切换用户等逻辑全部在服务器端实现，且数据在服务器端维护，保证数据一致性。

（8） 前台使用 curses 库进行编写，可以实时显示坦克状态与对战状态

（9） 系统使用了 itimer 库与 signal 库，itimer 库使“机器人”坦克能定时移动，发射子弹。Signal 库保证客户端能及时处理服务器传来的消息。

（10） 游戏引入了切换用户功能,用户可以通过切换,操作两个坦克(主体坦克与机器人)，因为机器人坦克被攻击时不会掉血，因此该功能提高了用户生存率。

（11） 游戏设置了超时检测，将网络状况不好，长时间（60s）没有配对的客户端及时踢掉，关闭对应 socket，避免占用服务器资源。

（12） 定义了协议格式与消息解析方式，且传输数据少，基本只涉及到房间号与队伍号，保证游戏的实时性。

功能：

（1） 服务器可以支持多局二人对战的并发

基本流程：

服务器设计了 room 结构体，为每两个对战用户专门开辟一个房间，这样，服务器端对坦克操作都是基于房间内的两个对战坦克的操作，服务器可以同时支持多局二人对战。

（2） 服务器端为每个房间随机生成一张地图，发送过客户端

基本流程：

两个对战用户就绪后，服务器调用 init 方法随机生成地图，障碍物。并将整张随机产生成的地图发送给两个客户端，客户端接受随机地图后显示，保证每个房间内对战双方地图一致，而不同房间互相地图不同。

（3） 用户坦克的基本行进

基本流程：

用户输入指令，客户端将指令发送给服务器端，服务器端收到对应指令后，更新服务器端记录的坦克位置，将数据广播给该游戏房间里的两个用户。

（4） 用户坦克的开火

基本流程：

用户输入指令，客户端将指令发送给服务器端，服务器端收到对应指令后，根据坦克位置与方向，新建子弹对象，加入子弹 vector 中，将数据发送给客户端，客户端收到信息后，同样新建子弹对象，加入子弹 vector 中，在前台不断遍历此 vector，显示子弹轨迹。

（5） “自动机器人”坦克的行进

基本流程：

前端使用 itimer，定时每 7 秒，客户端向服务器请求给房间的机器人状态更新（不设计为服务器自动发送因为服务器需要知道是要给哪个房间更新，二这一信息必须由客户端传递）。服务器收到指令后，更新机器人状态，将新的信息广播给该房间内两个用户，用户根据收到的信息，解析更新状态。

（6） “自动机器人”坦克的开火

基本流程：

每七秒前端收到更新状态的指令，客户端根据状态中的“shoot”选项值，判断是否应该开火，若机器人坦克需要开火，则采用与用户坦克开火相似的协议进行传输，协议内各个数据不变，不过协议标志位需要更改，以便服务器知道是机器人坦克开的火，服务器处理后将对应子弹状态发送给客户端，客户端根据对应信息设置子弹状态，达到队友之间子弹不伤害，对手之间子弹对主体坦克产生伤害并掉血，对应结果计入我方积分，对机器人坦克只做被击中后的复活处理，不会造成伤害，也不计入我方积分，这一过程通过为子弹新增“enemyshoot”字段实现，击中时对这一子弹判断即可。

（7） 血量与得分之间的转化

基本流程：

每七秒前端收到更新状态的指令，客户端根据状态中的“shoot”选项值，判断是否应该开火，若机器人坦克需要开火，则采用与用户坦克开火相似的协议进行传输，协议内各个数据不变，不过协议标志位需要更改，以便服务器知道是机器人坦克开的火，服务器处理后将对应子弹状态发送给客户端，客户端根据对应信息设置子弹状态，达到队友之间子弹不伤害，对手之间子弹对主体坦克产生伤害并掉血，对应结果计入我方积分，对机器人坦克只做被击中后的复活处理，不会造成伤害，也不计入我方积分，这一过程通过为子弹新增“enemyshoot”字段实现，击中时对这一子弹判断即可。

（8） 用户坦克与“自动机器人”坦克之间的切换

基本流程：

本游戏中设计了机器人坦克这一角色，每个主体坦克带着一个机器人坦克，并设计了切换角色功能，用户按下 l 键，可将两个坦克切换，这一功能通过维护全局变量的方式实现，客户端维护了 palyer[1]，player[2]，enemy[1]，enemy[2]，

服务器端维护了 player[1]，player[2]，player[3]，player[4]。对应下标中的 1,2..使用全局变量维护如下：

客户端：

int p0=0;　int e0=0;　int p1=1;　int e1=1

服务器：

int p0=0;　int p1=1;　int h0=2;　int h1=3;

所有的操作都使用全局变量进行维护，当切换时服务器端将对应 player 信息置换，再将对应全局变量交换，之后发送信息给客户端，客户端根据协议判断应该交换 player（玩家）还是 enemy（敌人），之后对应交换，显示，达到同步的效果。

（9） 游戏逻辑以及子弹的形状设计

在服务器端，维护了游戏的整体逻辑，包括坦克的碰撞检测，以及坦克移动，开火，切换的逻辑，并维护了游戏的全部数据。客户端维护了各种显示的信息，此外，为了使子弹显示清晰，我设计了两种子弹形状“o”与“x”，其中 o 表示使射向对方的子弹，而 x 表示队友射出的子弹，方便用户判断需不需要躲避子弹。这一逻辑通过为“开火”协议设置相应“enemyshoot”字段来实现。

（10） “大本营”的设计

在本游戏中，杀死对手除了将对手主战坦克声明值打为 0 之外，还可以通过摧毁对方大本营的方式达到这一目的。实现逻辑：服务器判断击中位置，若地图上击中位置为“$”，则根据协议获取当前用户，之后当前用户的对手判定为输，当前用户判定为赢，将信息广播给两个用户，对应客户端接受信息，处理。

（11） 复活设计：

主体坦克，机器人坦克在复活点进行复活，切换用户后，主体坦克（原机器人坦克）在原主体坦克处复活，机器人坦克（原主体坦克）在原主体坦克处复活。

（12） 游戏房间的设计

本游戏支持多组二对二游戏，服务器为每两个用户分配了房间，即第一个进入的是房间 1 的 a 队，第二个进入的是房间 1 的 b 队，第三个进入的是房间 2 的 a 队…以此类推。因此，服务器端定义了 room 结构体，结构如下：

```
struct room{
    int fd1;
    int fd2;
    int roomnum;
    int fd1life;
    int fd1points;
    int fd2life;
    int fd2points;
    Tank player[4];
    char warfield[Fieldy][Fieldx];
        }rooms[30];
```

room 结构体中定义了对战双方的基本信息，包括 fd，点数，生命等信息。之后服务器队坦克的基本逻辑操作都基于房间对象进行操作，客户端将房间号，队号发送给服务器，服务器得到对应信息后进行逻辑处理之后将对应信息发送回给客户端显示。

（13） 超时自动剔除资源设计

当一个用户进入房间后，由于网络原因迟迟没有用户与之配对，则不应再让他

占用服务器资源,这一设计结合了 epoll 的事件触发机制,在服务器端定义了 event 结构体，其中维护了时间字段，当用户有操作时，更新时间为当前时间。如长时间这一字段不变，则将对应 socket 关闭，并发送提示文字。本系统中设计该时间为 60s。

（14）用户协议设计

服务器端协议发送：

1. 基本移动：

   w/a/s/d |队伍号|房间号

2. 攻击：

   f|队伍号|房间号

3. 回血：

   o|队伍号|房间号

4. 切换用户：

   changeok|队伍号|房间号

5. 主体坦克被击中：

   h|队伍号|房间号

6. “机器人”坦克被击中：

   hai/hbi|房间号

7. 大本营被击中：

   home|队伍号|房间号

8. 更新机器人坦克状态：

   ghmp|房间号

9. 超时链接：

   timeout|房间号

客户端协议解析：

10.

标志位|x 值（可选）|y 值（可选）|direction（可选）|shoot（金 ghmp 指令解析）|ident|

- 其中：

  x，y，direction 值在不同机制中有不同含义，shoot 值仅在 ghmp 指令中指定机器人坦克是否开火,ident 值判断传来的指令是用来改变该客户端的 enemy 的还是改变

该客户端的 player 的。

（15） 客户端消息接受机制设计

客户端拥有两个进程，子进程负责游戏逻辑的处理，界面的显示，父进程负责一直 read 服务器端传来的信息，当收到了服务器传来的信息后，通过信号机制，将信号发送给子进程，通知子进程有任务处理，调用相应函数。之后将信息写入管道，子进程收到信号后，从管道中读取相应数据，根据协议，进行处理。

（16） 服务器端反应堆模型设计

服务器端设计参考了开源网络库 libevent 的设计模式，采用事件-回调函数机制对产生的任务队列进行快速处理。

（17） 服务器端异步 I/O：epoll 设计

服务器端异步 I/O 采用 epoll 方式实现，保证以事件驱动的方式有序处理各类任务，且效率更高，避免了 select 的遍历操作。

（18） 客户端的父子进程通信

客户端父子进程之间通过 siguser1 与 siguser2 两个信号实现父子进程之间的通信。其中 siguser1 处理用户死亡后的逻辑，siguser2 负责处理基本的服务器端传输协议后对应反应逻辑。

（19） 同步机制的运用

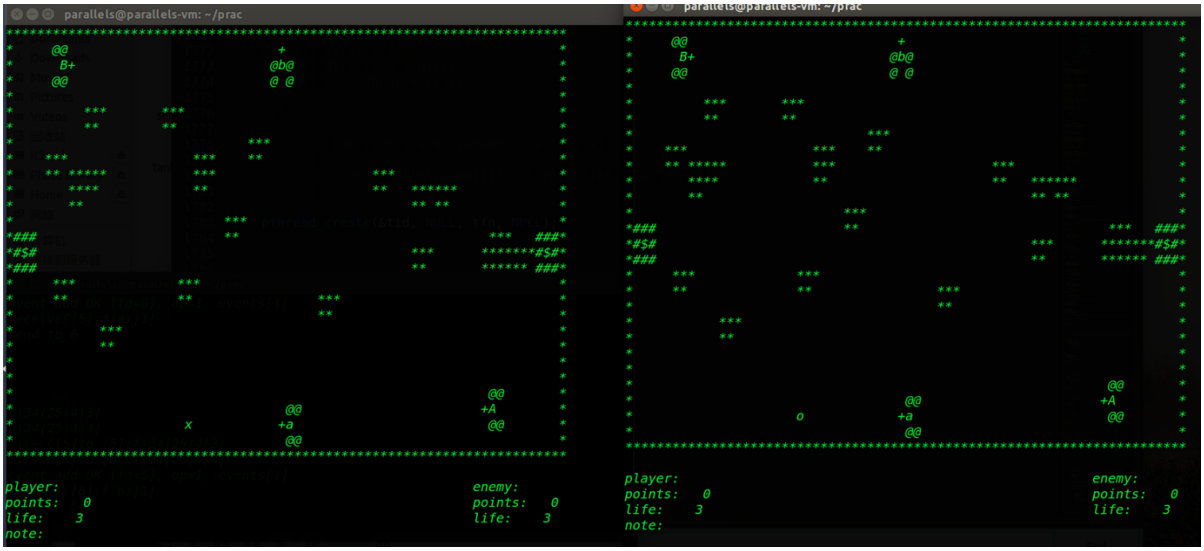
在用户发射子弹时，因为我使用了 vector 对子弹对象进行存储，其 pushback 方法速度较慢，因此我单开一个线程对子弹进行添加，并将子弹维护为全局变量。这一步骤加入了 mutex 机制。以此类推，前台对子弹的消失，遍历显示等操作也是采用同样操作，实现同步机制。

（20） 坦克之间碰撞检测功能设计

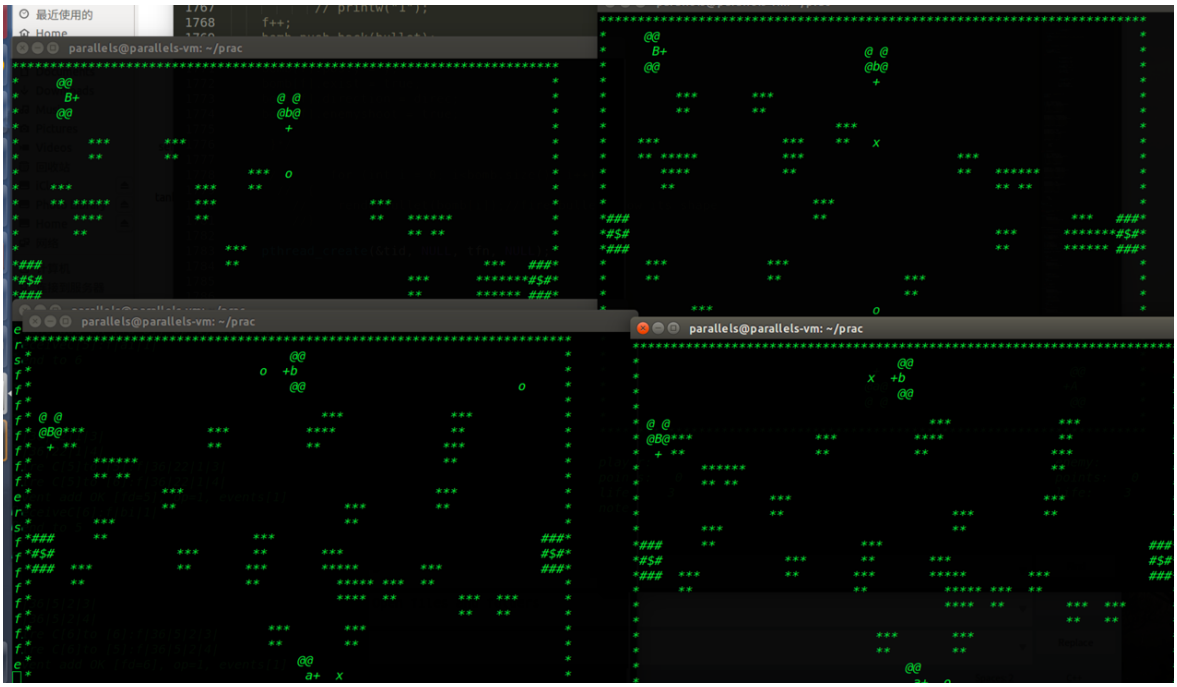
在服务器端，坦克移动后需要对坦克位置进行碰撞检测，一旦坦克与其他坦克有碰撞，则将坦克位置回退到之前位置，将这一位置发送给客户端，避免坦克之间的重叠。

四． 基本展示

1. 双人对战界面：



2. **在同一服务器上的多人同时游戏，两两一个房间，房间内数据同步，房间之间数据互不影响**



3. 用户切换功能简介：

最初默认用户操作主战坦克 A（左侧），a（右侧）坦克自动变向，自动开火。当用户切换主战坦克后，用户可操作 a，此时 A 坦克自动变向，自动开火，被击中不计入失血。

4. 积分界面：

五． 实验感想与收获

在本次实验中，我学会了 Linux 编程的相关知识，并进一步认识了操作系统的基本知识，同时，对小实验中的相关管道，进程，线程，同步等问题有了基本认识，此外，本次实验中我还自学到了很多例如 epoll 等机制的原理，我还参考了开源网络库 libevent 的事件驱动模式，使用反应堆模型对接受到的信息通过回调函数及时处理，优化了性能。但是，本游戏现在还有一些功能不够完善，需要进一步改进。在整个大实验的编写过程中，我的编程能力得到了很大的提升，也为未来职业道路打下了坚实的基础。