任务一

```
pore@localhost:~/lab9/task/roshambo$ make
gcc -Wall -O3 -fPIC -shared roshambo_cheater.c -o roshambo_cheater.so -ldl
pore@localhost:~/lab9/task/roshambo$ ls
makefile  roshambo  roshambo_cheater.c  roshambo_cheater.so
```

Rand()函数主要用于生成伪随机数，在使用 rand()函数之前，通常通过 srand()

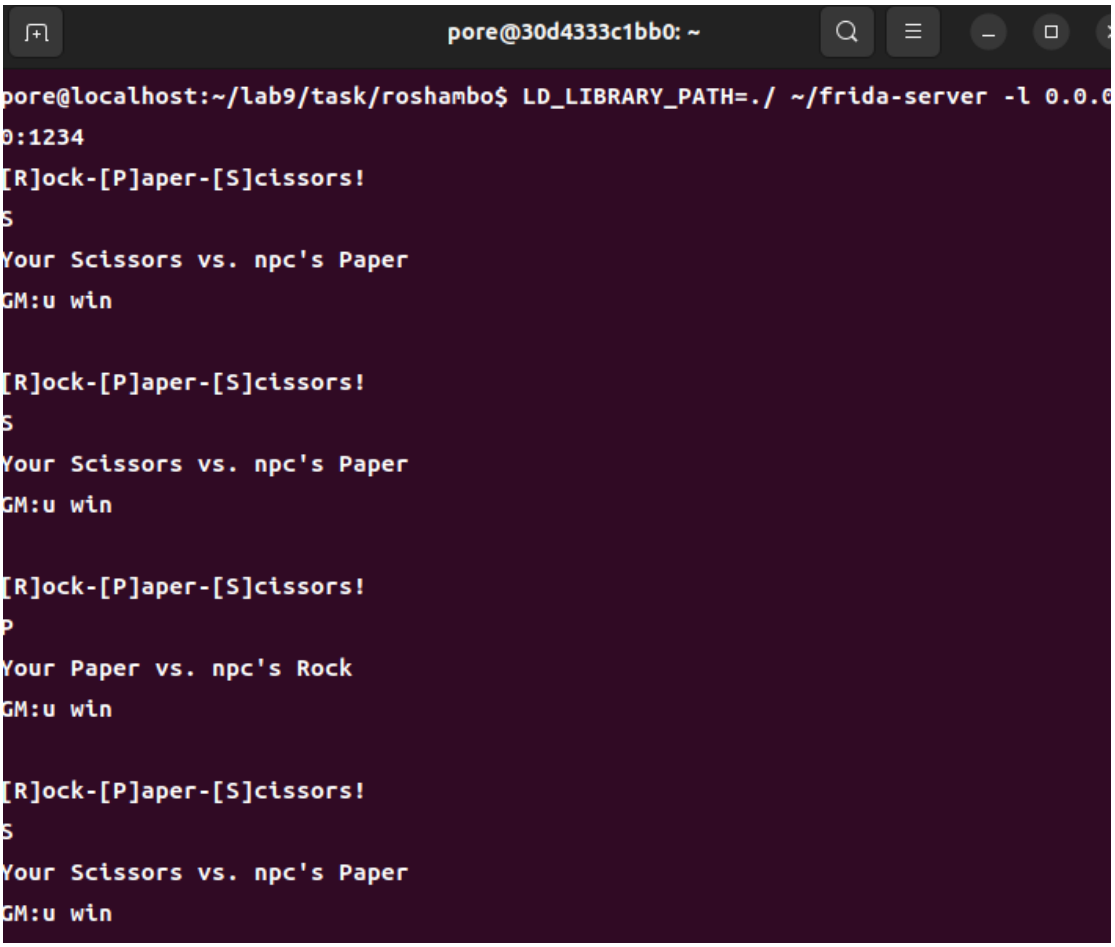
函数来设置随机数生成器的种子值。

```
pore@localhost:~/lab9/task/roshambo$ LD_PRELOAD=./roshambo_cheater.so ./roshambo
[R]ock-[P]aper-[S]cissors!
P
Your Paper vs. npc's Rock
GM:u win

[R]ock-[P]aper-[S]cissors!
P
Your Paper vs. npc's Rock
GM:u win

[R]ock-[P]aper-[S]cissors!
P
Your Paper vs. npc's Rock
GM:u win
```

```c
//roshambo_cheater.c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>


int rand(void) {
    void *handle = dlopen("libc.so.6", RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, "无法打开库: %s\n", dlerror());
        exit(1);
    }
    dlerror();

    // 关闭动态链接库
    dlclose(handle);

    return 0;
}
```

将原始的 rand 函数换成带有辅助功能的函数，让函数直接返回零，那么对方出的一致都是拳头，玩家只要出布就能获胜。

任务二


pore@30d4333c1bb0: ~
pore@localhost:~/lab9/task/roshambo$ LD_LIBRARY_PATH=./ ~/frida-server -l 0.0.0
0:1234
[R]ock-[P]aper-[S]cissors!
S
Your Scissors vs. npc's Paper
GM:u win
[R]ock-[P]aper-[S]cissors!
S
Your Scissors vs. npc's Paper
GM:u win
[R]ock-[P]aper-[S]cissors!
P
Your Paper vs. npc's Rock
GM:u win
[R]ock-[P]aper-[S]cissors!
S
Your Scissors vs. npc's Paper
GM:u win

```
var moduleName = "libc.so.6";
var functionName = "srand";

// 寻找目标函数的地址
var funcAddress = Module.findExportByName(moduleName, functionN
if (funcAddress === null) {
    throw new Error('Function not found.');
}
console.log(functionName + " function address: " + funcAddress

// 插桩目标函数srand
Interceptor.attach(funcAddress, {
    onEnter: function(args) {
        // 输出原始参数
        console.log("Original argument: " + args[0]);
        // 修改原始参数
        args[0] = ptr("0");

    },
    // 当函数返回时，此代码块执行
    /* onLeave: function(retval) {
        if (doit == true){
            // 输出原始返回值
            console.log("Original return value: " + retval);

            // 修改返回值
            var newRetval = ptr("0"); //
            retval.replace(newRetval);

            console.log("Return value changed to: " + newRetva
            doit == false;
        }
    } */
});
```

Srand 函数不需要返回值，所以不考虑返回的情况

srand 的参数实际上是一个种子(seed)，相当于告诉计算机从哪里开始生成随机数序列。如果使用相同的种子，那么每次程序运行时生成的随机数序列都会是相同的。直接指定固定的参数 0，使得对手的出法每一轮游戏都会相同。在这个参属下，对手固定出 PPRPSPP……玩家只需要依次出 SSPSRSS……就能取胜。

任务三

因为 open、unlink 操作需要将参数转换为字符串并输出的功能，所以在一起处理。Write 和 read 单独处理。

```
// 定义需要插桩的文件操作相关的系统调用
var fileApiFunctions = ["open","unlink"];

// 针对open、unlink进行插桩
fileApiFunctions.forEach(function (funcName) {
    var funcAddress = Module.findExportByName(null, funcName);
    if (funcAddress !== null) {
        Interceptor.attach(funcAddress, {
            onEnter: function (args) {
                // 输出系统调用及参数
                console.log("[*] Function: " + funcName);
                // 检查参数是否为有效地址
                if (args[0].isNull()) {
                    console.log("Invalid address detected. Skipping operation.");
                    return;
                } else {
                    console.log("Open original argument: " + args[0].readUtf8String());
                }
            }
        });
    } else {
        console.log("Function not found: " + funcName);
    }
});
```

```
// 对写操作进行额外插桩处理
Interceptor.attach(Module.findExportByName(null, 'write'), {
    onEnter: function(args) {
        // 输出原始参数
        var fd = args[0];
        var buf = args[1];
        var count = args[2];
        console.log("[*] Function: " + "write");
        console.log("Open original argument: " + args[0]);
        console.log("Write: fd=" + fd + ", buf=" + buf + ", count=" + count);
    },
});

// 对读操作进行额外插桩处理
Interceptor.attach(Module.findExportByName(null, 'read'), {
    onEnter: function(args) {
        // 输出原始参数
        var fd = args[0];
        var buf = args[1];
        var count = args[2];
        console.log("[*] Function: " + "read");
        console.log("Open original argument: " + args[0]);
        console.log("Read: fd=" + fd + ", buf=" + buf + ", count=" + count);
    },
});
```

脚本输出得到的结果如图：

```
[Remote::WannaMedia.naive ]-> [*] Function: open
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted
Read: fd=0x6, buf=0xaaaad47cfdb0, count=0x3c
[*] Function: open
Open original argument: /dev/null
Write: fd=0x6, buf=0xffffda5a7428, count=0x1000
[*] Function: unlink
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted
[*] Function: open
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted.encryp
ted
Write: fd=0x6, buf=0xaaaad47cfe00, count=0x3c
[*] Function: open
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted.encryp
ted.encrypted.encrypted
Read: fd=0x6, buf=0xaaaad47cfdb0, count=0x3b
[*] Function: open
Open original argument: /dev/null
Write: fd=0x6, buf=0xffffda5a7428, count=0x1000
[*] Function: unlink
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted.encryp
ted.encrypted.encrypted
[*] Function: open
Open original argument: /tmp/media/lab9.txt.encrypted.encrypted.encrypted.encryp
ted.encrypted.encrypted.encrypted
Write: fd=0x6, buf=0xaaaad47cfe00, count=0x3c
```

第一个文件交互阶段:
打开文件 /tmp/media/lab9.txt.encrypted.encrypted.encrypted 并进行读取操作，读取了 0x3c 字节的数据。

第二个文件交互阶段:
打开 /dev/null 这个特殊的设备文件，并向其写入了 0x1000 字节的数据。

第三个文件交互阶段:
使用 unlink 操作删除文件 /tmp/media/lab9.txt.encrypted.encrypted.encrypted。

第四个文件交互阶段:
打开了文件 /tmp/media/lab9.txt.encrypted.encrypted.encrypted.encrypted，并向该文件写入了 0x3c 字节的数据。

……
Wannamedia 使用了多个嵌套的加密后缀命名文件来读取，并且多次重复写入数据，可能是为了混淆。
使用 /dev/null 这个特殊的设备文件进行写入操作，这可能是为了隐藏数据。
使用 unlink 删除文件，可能是为了隐藏操作痕迹。