

沈阳航空航天大学

# 课 程 设 计 报 告

课程设计名称：操作系统课程设计

课程设计题目：Linux 系统进程间通信一  
利用管道实现

学 院：计算机学院

指导教师：董燕举

学生信息：何雨泊、杨晨、汪恒辉

专业	班级	学号	姓名
软件工程	软件1801	183401050120	何雨泊
软件工程	软件1801	183401050121	杨晨
软件工程	软件1801	183401050118	汪恒辉

完成时间：2020年7月8日

## 沈阳航空航天大学

## 课程设计任务书

课程设计名称	操作系统课程设计		
题目名称	Linux 系统进程间通信—利用管道实现		
起止日期	2020 年 6 月 22 日起 至 2020 年 7 月 3 日止		
学生信息			
专业	班级	学号	姓名
软件工程	软件 1801	183401050120	何雨泊
软件工程	软件 1801	183401050121	杨晨
软件工程	软件 1801	183401050118	汪恒辉
课设内容和要求：			
<p>编写一个程序，用 Linux 中的 IPC 机制中的管道完成两个进程进行石头、剪子、布的游戏。</p> <p>可以创建三个进程，其中，一个进程为裁判进程，另外两个进程为选手进程。可以将石头、剪子、布这三招定义为三个整型值。胜负关系：石头〉剪子〉布〉石头。选手进程按照某种策略（例如，随机产生）出招，交给裁判进程判断大小。裁判进程将对手的出招和胜负结果通知选手。比赛可以采取多盘（&gt;100 盘）定胜负，由裁判宣布最后结果。每次出招由裁判限定时间，超时判负。 每盘结果可以存放在文件或其他数据结构中。比赛结束，可以打印每盘的胜负情况和总的结果。</p> <p>所谓管道，是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称 pipe 文件。向管道（共享文件）提供输入的发送进程（即写进程），以字符流形式将大量的数据送入管道，而接收管道输出的接收进程（即读进程），可从管道中接收数据。由于发送进程和接收进程是利用管道进行通信的，故又称管道通信。这种方式能够有效传送大量数据，而且管理最适合在进程之间实现生产者/消费者的交互。有些进程往管道中写入数据，而另外一些进程则从管道中读出数据。</p> <p>为了协调双方的通信，管道通信机制必须提供以下三方面的协调能力：</p>			

- (1) 互斥。当一个进程正在对 pipe 进行读/写操作时，另一个进程必须等待；
- (2) 同步。当写（输入）进程把一定数量数据写入 pipe 后，便去睡眠等待，直到读（输出）进程取走数据后，再把它唤醒。当读进程读到一空 pipe 时，也应睡眠等待，直至写进程将数据写入管道后，才将它唤醒；
- (3) 对方是否存在。只有确定对方已存在时，才能进行通信。

**要求：**

- 1、 设计表示“石头、剪子、布”的数据结构，以及它们之间的大小规则；
  - 2、 设计比赛结果的存放方式；
  - 3、 完成选手进程设计；
- 完成裁判进程设计。

**参考资料：**

- [1] 北京英真时代. Linux 内核实验教程\_2.1. 北京英真时代科技有限公司编写
- [2] 赵炯. Linux 内核完全注释. 北京:机械工业出版社, 2004
- [3] 赵炯. Linux 内核完全剖析——基于 0.12 内核. 北京: 机械工业出版社, 2008
- [4] 拉芙(RobertLove).Linux 内核设计与实现(第 3 版). 北京: 机械工业出版社, 2011
- [5] 汤小丹等. 计算机操作系统（第 3 版）. 西安电子科技大学出版社, 2007.05

系 审 核 意 见： 同 意 ☒ 不 同 意 ☐ 系 主 任 签 字（ 盖 章 ）：

指导教师（签名） 2020 年 6 月 19 日

学 生（签名） 何雨泊 2020 年 6 月 22 日

学 生（签名） 杨晨 2020 年 6 月 22 日

学 生（签名） 汪恒辉 2020 年 6 月 22 日

## 目 录

<b>1</b>	<b>项目分工与实施计划.....</b>	<b>1</b>
1.1	问题描述.....	1
1.2	项目分工.....	3
1.2.1	成员简介.....	3
1.2.2	系统开发分工.....	3
1.2.3	文档撰写分工.....	3
1.3	开发计划.....	4
<b>2</b>	<b>理论概述与源代码分析.....</b>	<b>5</b>
2.1	理论概述.....	5
2.1.1	系统调用相关原理.....	5
2.1.2	Linux 内核相关原理.....	5
2.1.3	进程的创建.....	6
2.1.4	调度进程.....	6
2.1.5	管道通信.....	7
2.1.6	信号量.....	7
2.2	源代码分析.....	8
2.2.1	源代码文件和宏定义.....	8
2.2.2	函数功能及其描述.....	9
<b>3</b>	<b>算法及数据结构设计.....</b>	<b>14</b>
3.1	数据结构设计.....	14
3.2	算法设计.....	17
3.2.1	程序整体算法设计.....	17
3.2.2	选手 1 出拳算法设计.....	18
3.2.3	选手 2 出拳算法设计.....	18
3.2.4	裁判判定设计.....	19
3.2.5	出拳时间算法设计.....	20
3.2.6	信号量并发控制算法设计.....	21
<b>4</b>	<b>程序测试与验证.....</b>	<b>23</b>
4.1	测试分析.....	23
4.2	测试结果.....	23
4.2.1	总体程序测试.....	23
4.2.1	选手出拳算法测试.....	27
<b>5</b>	<b>结论.....</b>	<b>30</b>
	<b>参考文献.....</b>	<b>31</b>
	<b>附 录 程序清单.....</b>	<b>32</b>

# 1 项目分工与实施计划

## 1.1 问题描述

Linux 环境下,进程地址空间相互独立,每个进程各自有不同的用户地址空间。任何一个进程的全局变量在另一个进程中都看不到,所以进程和进程之间不能相互访问,要交换数据必须通过内核,在内核中开辟一块缓冲区,进程 1 把数据从用户空间拷到内核缓冲区,进程 2 再从内核缓冲区把数据读走,内核提供的这种机制称为进程间通信(IPC)。

在该课题中,应创建三个进程,分别为:选手一的进程、选手二的进程以及裁判的进程,这三个进程之间通过管道来通信。对于选手而言,可以通过某种方式出招(如:预测对手的本次出拳、随机产生等等),并且将出招结果交由裁判;而裁判要根据选手一和选手二的出拳结果判断大小并决定胜负,与此同时,裁判也应将对手的出招以及胜负的结果分别通知给两位选手。对于石头、剪子、布这三招,应将其定义为三个整型值。该游戏的规则为:石头>剪子>布>石头,游戏有限定时间,如某一方出拳超时,则被判定为失败;如两方均出拳超时,则平局。比赛可以采取多盘(>100 盘)定胜负,由裁判宣布最后结果。每一局的结果可以存放在文件或其他的数据结构中,待比赛结束之后,应将每一局的胜负情况以及总结果加以表示。

进程之间的信息交换成为进程通信,而进程通信有很多种类型,管道通信就是其中的一种。所谓管道,是指用于连接一个读进程和一个写进程,以实现它们之间通信的共享文件,又称 pipe 文件。向管道(共享文件)提供输入的发送进程(即写进程),以字符流形式将大量的数据送入管道,而接收管道输出的接收进程(即读进程),可从管道中接收数据。由于发送进程和接收进程是利用管道进行通信的,故又称管道通信。

为了实现三个进程的同步控制,又要运用信号量。信号量本质上是一个计数器(不设置全局变量是因为进程间是相互独立的,而这不一定能看到,看到也不能保证引用计数为原子操作),用于多进程对共享数据对象的读取,它和管道有所

不同，它不以传送数据为主要目的，它主要是用来保护共享资源（信号量也属于临界资源），使得资源在一个时刻只有一个进程独享。

课题开发过程可描述如下：

#### （1）管道通讯机制的理解

所谓管道，是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称 pipe 文件。向管道（共享文件）提供输入的发送进程（即写进程），以字符流形式将大量的数据送入管道，而接收管道输出的接收进程（即读进程），可从管道中接收数据。由于发送进程和接收进程是利用管道进行通信的，故又称管道通信。

#### （2）数据结构设计

该题目的数据结构主要为选手的出招和每盘胜负情况的数据结构的设计，每盘的结果可以存放在文件或其他数据结构中。比赛结束，可以打印每盘的胜负情况和总的结果。

#### （3）算法流程设计

在此任务中要明确石头剪刀布选手出拳流程和原理以及结果判断的方式、如何用信号量实现进程的同步控制等。

#### （4）代码编写

程序的各功能模块可以在 Engintime Linux Lab 或其他代码编辑器中编写，然后将代码在 Engintime Linux Lab 中运行即可看到运行效果。

#### （5）测试调试

上一任务结束后，虽然在 Linux 中实现了裁判判断，但无法判断能否正常运行或是否存在 bug，所以需要设计测试程序来测试此调度算法。测试包括进程的 I/O 能否正常运行、睡眠进程能否正常唤醒以及低优先权进程能否提权。

#### （6）完善算法

根据测试和分析的结果对算法进行完善。

#### （7）撰写报告

总结和撰写报告。

## 1.2 项目分工

### 1.2.1 成员简介

描述小组每个成员的性格、兴趣、爱好和技术特长等。样例如下：

（1）何雨泊（组长）：熟练掌握数据结构相关算法，能较快学习理解 Linux 的相关知识，并实现一定功能。

（2）杨晨：拥有较强的 C 语言编写能力，可以实现特定功能，如此任务中的分析程序。

（3）汪恒辉：拥有较强的组织能力，文档编写能力，负责此任务相关文档的保存、整理与编写。

### 1.2.2 系统开发分工

（1）何雨泊：建立 Linux 内核项目，然后针对问题编辑相应的算法，对课程设计代码进行总体框架的设计，使项目有了大致的方向。实现三个进程之间同步互斥以及利用管道进行通信的功能并实现了对石头剪刀布游戏的数据结构的开发设计。

（2）杨晨：完成裁判进程的程序编写，设计并开发选手 1 和选手 2 的出拳算法。利用控制变量法对选手 1 和选手 2 的算法进行测试，同时完成延时毫秒级函数的编写，并对程序的相关功能进行测试。

（3）汪恒辉：完成信号量系统调用函数的添加和测试，使用信号量机制控制裁判和两个选手进程的并发执行，将每一轮选手之间的出拳和胜负情况以及比赛结束以后总成绩和胜负写入文本文件中。

### 1.2.3 文档撰写分工

表 1.1 文档撰写分工表

	何雨泊	杨晨	汪恒辉
项目实施计划与分工	√		
理论概述		√	
源代码分析			√
数据结构设计		√	
算法的分析与验证	√		√

结论			√
附录	√	√	
课程设计总结与体会	√	√	√

### 1.3 开发计划

(1) 6月23日至6月26日: 阅读相关书籍和Linux管道的资料, 理解Linux管道的通信机制与信号量。

(2) 6月25日至6月26日: 实现石头剪刀布游戏和系统所用信号量和管道的数据结构开发设计, 编写判断石头剪刀布游戏的函数代码, 完成裁判和两个选手进程代码的编写, 可以进行无序的石头剪刀布游戏。

(3) 6月27日: 通过查阅资料和百度, 学习C语言在Linux0.11环境下读写文件的操作, 使程序可以将每盘的运行结果和总的胜负情况存储在文本文件中。

(4) 6月28日至6月29日: 查询相关信号量并发控制的相关博客并完成实验七信号量的使用。

(5) 6月29日至7月3日: 将信号量相关代码加入总代码中, 实现利用信号量控制选手进程和裁判进程之间的并发控制, 可以完成实验要求的程序;

(6) 7月4日至7月5日: 分析差异, 修改函数的代码, 使程序可以利用Linux中的IPC机制中的管道完成游戏。

(7) 7月6日至7月7日: 答辩, 对老师提出的问题进行整改, 撰写课设报告。

(8) 7月7日至7月10日: 提交课设报告和考核记录表。



## 2 理论概述与源代码分析

### 2.1 理论概述

#### 2.1.1 系统调用相关原理

系统调用是操作系统为应用程序提供的与内核进行交互的一组接口。通过这些接口，用户态应用程序的进程可以切换到内核态，由系统调用对应的内核函数代表该进程继续运行，从而可以访问操作系统维护的各种资源，实现应用程序与内核的交互。系统调用通过中断机制向内核提交请求，系统调用的功能由内核函数实现，进入内核后不同的系统调用会有其各自对应的内核函数，这些内核函数就是系统调用的“服务例程”。Linux 内核为了区分不同的系统调用，为每个系统调用分配了唯一的系统调用号。这些系统调用号定义在文件 `include/unistd.h` 中，系统调用号实际上对应于 `include/linux/sys.h` 中定义的系统调用函数指针表数组 `sys_call_table[]` 中项的索引值（也就是数组的下标）。

#### 2.1.2 Linux 内核相关原理

Linux 内核主要由五个模块构成，他们分别是：进程调度模块、内存管理模块、文件系统模块、进程间通信模块和网络接口模块。其中进程调度模块用来负责控制进程对 CPU 资源的使用。所采取的的调度策略是各进程能够公平合理地访问 CPU，同时保证内核能及时地执行硬件操作。

这几个模块之间的依赖关系如图所示。图中主要包括五个部分：进程调度、内存管理、进程间通信、网络接口以及虚拟文件系统和文件系统。进程调度是中心，其余四部分都与进程调度相连接，其中特殊的是内存管理与进程调度可以相互转换。连线代表它们之间的依赖关系，虚线和虚框部分表示 Linux0.11 中还未实现的部分。

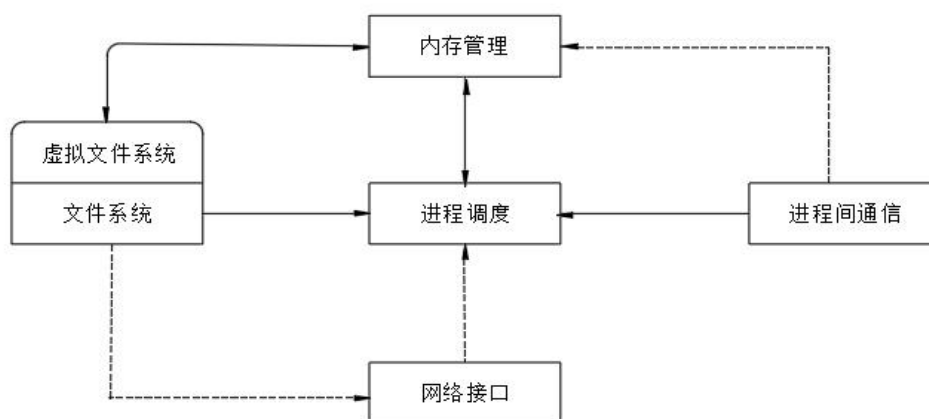


图 2.1 Linux 模块依赖关系图

### 2.1.3 进程的创建

程序通常是指一个可执行文件，而进程则是一个正在执行的程序实例。利用分时技术，在 Linux 操作系统上可以同时运行多个进程。分时技术的基本原理是把 CPU 的运行时间分成一个个规定长度的时间片，让每个进程在一个时间片内运行。当一个进程的时间片用完时，操作系统就利用调度程序切换到另一个进程去运行。因此，本质上对于具有单个 CPU 的机器来说，某一时刻只有一个进程在运行，但是由于进程运行的时间片比较短（通常为几十毫秒），所以给用户的感觉是多个进程在同时运行。

### 2.1.4 调度进程

当一个进程的时间片用完时，操作系统会使用调度程序强制切换到其他的进程去执行。另外，如果进程在内核态执行时需要等待系统的某个资源，此时该进程就会调用 `sleep_on` 或 `interruptible_sleep_on` 函数自愿地放弃 CPU 的使用权，从而让调度程序去执行其他进程，此进程则进入睡眠状态（`TASK_UNINTERRUPTIBLE` 或 `TASK_INTERRUPTIBLE`）。只有当进程从“内核运行态”转换到“睡眠状态”时，内核才会进行进程切换操作。在内核态下运行的进程不能被其他进程抢占，而且一个进程不能改变另一个进程的状态。为了避免进程切换时造成内核数据错误，内核在执行临界区代码时会禁止一切中断。

### 2.1.5 管道通信

管道是一种最基本的 IPC 机制,作用于有血缘关系的进程之间,完成数据传递。调用 `pipe` 系统函数即可创建一个管道。管道有如下特质:

- (1) 其本质是一个伪文件(实为内核缓冲区)
- (2) 由两个文件描述符引用,一个表示读端,一个表示写端。
- (3) 规定数据从管道的写端流入管道,从读端流出。

管道读写方式如图 2.2 所示。写进程在管道的尾端写入数据,读进程在管道的首端读出数据。数据读出后将从管道中移走,其它读进程都不能再读到这些数据。而用户进程通过 `Fd[0]` 来读管道,通过 `Fd[1]` 来写管道。



图 2.2 管道读写方式示意图

管道提供了简单的流控制机制。进程试图读一个空管道时,在数据写入管道前,进程将一直阻塞。同样,管道已经满时,进程再试图写管道,在其它进程从管道中读走数据之前,写进程将一直阻塞。

### 2.1.6 信号量

信号量是描述资源可用性的计数器,信号量可以通过创建一个值为 1 的信号量来专门锁定某个对象,如果信号量的值大于零,则资源可用,进程分配“资源的一个单元”,信号量减少一个。在多个进程之间设置一个信号量 `sem_id`,当一个进程执行 `sem_wait` 时,先判断这个 `sem_id` 是否为 0,为 0 则 wait。另一个进程执行 `sem_post` 把 value 加一,这样如果有 `sem_wait` 的线程则检查到 `sem_id`

不再为 0，则返回。继续执行。这样就完成了多个进程之间的同步。在一个进程执行之前先 `sem_wait` 检查一下资源是否可用，可用则把信号量减一并继续执行，执行结束之后再 `sem_post` 一下，把信号量加一，这样可以唤醒另外一个正在等待资源的进程，. 不可用则等待其他进程释放资源。这样就实现了进程的同步。

## 2.2 源代码分析

### 2.2.1 源代码文件和宏定义

表 2.1 源代码文件表

源代码文件	功能描述
<code>string.h/stdio.h</code> <code>unistd.h</code>	定义输入输出函数，包含课设中所用的以提供你所需要的函数和一些变量的声明，例如 <code>pipe()</code> 创建管道函数，和一些操作管道的函数。
<code>kernel/fork.c</code>	包含了创建进程的主要源代码。
<code>time.h</code>	日期和时间头文件。用于需要时间方面的函数，主要应用于课设实验中的随机数的生成。
<code>sys/stat.h</code> <code>sys/types.h</code>	是 Unix/Linux 系统的基本系统数据类型的头文件，含有 <code>size_t</code> ， <code>time_t</code> ， <code>pid_t</code> 等类型。创建管道的必要头文件。
<code>design.c</code>	实现该课设的 c 程序文件，包含石头剪刀布和管道的数据结构设计、选手裁判进程的创建、信号量并发控制的游戏实现文件。
<code>semaphore.c</code>	信号量系统调用函数实现文件。

表 2.2 宏定义表

```
typedef void sem_t;
#define __NR_sem_open 87
#define __NR_sem_wait 88
#define __NR_sem_post 89
#define __NR_sem_unlink 90
_syscall2(int, sem_open, const char*, name, unsigned int, value)
_syscall1(int, sem_wait, sem_t *, sem)
_syscall1(int, sem_post, sem_t *, sem)
_syscall1(int, sem_unlink, const char *, name)
```

## 2.2.2 函数功能及其描述

### (1) fork()

函数原型: kernel/sys\_call.s: void \_sys\_fork(void)

返回值: void

功能概述: 此函数是 fork 系统调用, 用于从当前进程拷贝一个进程; 若已经没有有效的进程空位了则直接返回; 否则调用 copy\_process 函数拷贝进程数据

参数说明: 无

具体说明: fork 是一个系统调用函数。该系统调用函数在执行时, 会在进程表中创建一个与调用此函数的进程(父进程)几乎完全一样的新的进程表项(子进程), 子进程与父进程执行同样的代码, 但该子进程拥有自己的数据空间和环境参数。在 fork 函数的返回位置处, 父进程将恢复执行, 而子进程也从相同的位置开始执行。在父进程中, 调用 fork 返回的值是子进程的进程标识号 PID, 而在子进程中 fork 函数返回的值是 0 (这正是 fork 的神奇之处, 调用一次, 返回两次)。通常会在分支语句中使用 fork 函数的返回值, 从而使父进程和子进程开始运行不同的代码。

在父进程中可以调用 wait 函数或 waitpid 函数阻塞父进程, 直到子进程退出后才从这 2 个函数中返回。这两个函数的原型在 include/sys/wait.h 文件中定义如下:

```
pid_t wait( pid_t * wait_loc )
pid_t waitpid( pid_t pid, pid_t * wait_loc, int options )
```

### (2) pipe()

函数原型:

```
#include<unistd.h> int pipe(int fd[2])
```

返回值: 成功 0 失败 -1

功能概述: pipe() 会建立管道, 并将文件描述词由参数 filedes 数组返回。filedes[0] 为管道里的读取端, filedes[1] 则为管道的写入端。

参数说明: 函数传入值 fd[2]: 管道的两个文件描述符, 之后就是可以直接操作两个文件描述符。

具体说明: 管道是一种把两个进程之间的标准输入和标准输出连接起来的机

制，从而提供一种让多个进程间通信的方法，当进程创建管道时，每次都需要提供两个文件描述符来操作管道。其中一个对管道进行写操作，另一个对管道进行读操作。对管道的读写与一般的 IO 系统函数一致，使用 `write()` 函数写入数据，使用 `read()` 读出数据返回值：成功，返回 0，否则返回 -1。参数数组包含 pipe 使用的两个文件的描述符。`fd[0]`:读管道，`fd[1]`:写管道。

必须在 `fork()` 中调用 `pipe()`，否则子进程不会继承文件描述符。两个进程不共享祖先进程，就不能使用 pipe。

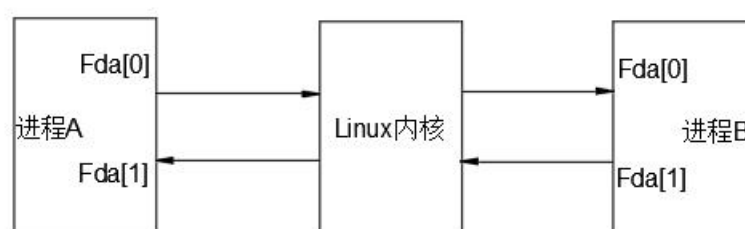


图 2.3 隐形管道的读取

结合本课设在一个选手进程和裁判进程之间建立两个管道，管道一用来存放选手一的出拳结果，管道二用来存放裁判的输赢和统计结果，故一共建立了四个管道，下面给出选手一和裁判之间的管道，如图 2.9 所示。

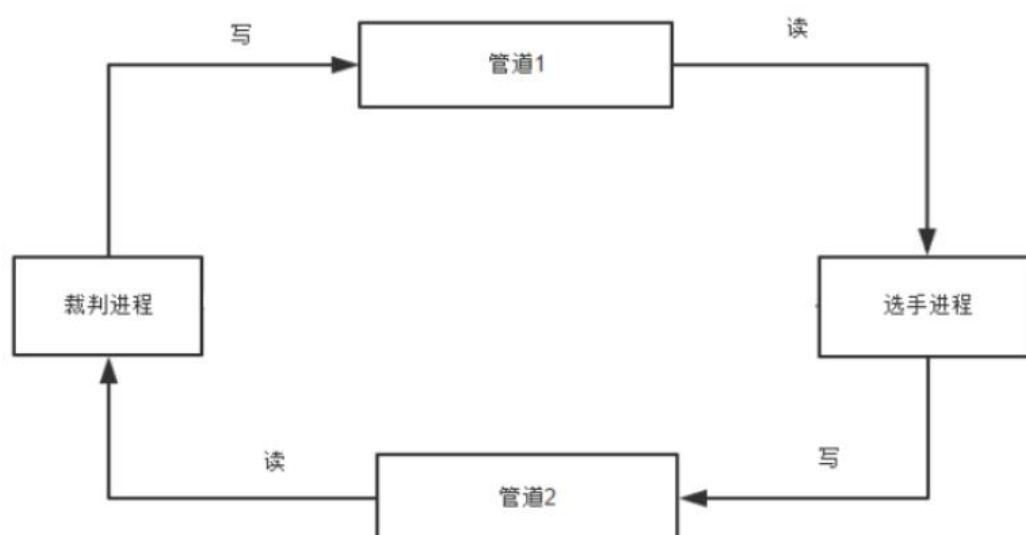


图 2.4 管道的建立及通信

### (3) `srand()`

函数原型: `void srand(unsigned int seed)`

返回值: `void`

功能概述: `srand` 函数是随机数发生器的初始化函数 `srand` 和 `rand()` 配合使用产生伪随机数序列

参数说明: 参数 `seed` 必须是个整数, 通常可以利用 `time(0)` 的返回值或 `NULL` 来当做 `seed`。

具体说明: `rand` 函数在产生随机数前, 需要系统提供的生成伪随机数序列的种子, `rand` 根据这个种子的值产生一系列随机数。如果系统提供的种子没有变化, 每次调用 `rand` 函数生成的伪随机数序列都是一样的。`srand(unsigned seed)` 通过参数 `seed` 改变系统提供的种子值, 从而可以使得每次调用 `rand` 函数生成的伪随机数序列不同, 从而实现真正意义上的“随机”。通常可以利用系统时间来改变系统的种子值, 即 `srand(time(NULL))`, 可以为 `rand` 函数提供不同的种子值, 进而产生不同的随机数序列。

#### (4) `rand()`

函数原型: `int rand(void)`

返回值: `void`

功能概述: `rand` 函数不是真正的随机数生成器, 而 `srand()` 会设置供 `rand()` 使用的随机数种子。如果你在第一次调用 `rand()` 之前没有调用 `srand()`, 那么系统会为你自动调用 `srand()`。而使用同种子相同的数调用 `rand()` 会导致相同的随机数序列被生成。

参数说明: `null`

具体说明: `srand((unsigned)time(NULL))` 则使用系统定时/计数器的值作为随机种子。每个种子对应一组根据算法预先生成的随机数, 所以, 在相同的平台环境下, 不同时间产生的随机数会是不同的, 相应的, 若将 `srand(unsigned)time(NULL)` 改为 `srand(TP)` (`TP` 为任一常量), 则无论何时运行、运行多少次得到的“随机数”都会是一组固定的序列, 因此 `srand` 生成的随机数是伪随机数。

#### (5) `sem_open()`

函数原型: `int sem_open(const char* name, unsigned int value)`

返回值：当成功时，返回值是该信号量的唯一标识（比如，在内核的地址、ID 等）。如失败，返回值是 NULL。

功能概述：创建一个信号量，或打开一个已经存在的信号量。

参数说明：

name, 信号量的名字。不同的进程可以通过同样的 name 而共享同一个信号量。如果该信号量不存在，就创建新的名为 name 的信号量；如果存在，就打开已经存在的名为 name 的信号量。

value, 信号量的初值，仅当新建信号量时，此参数才有效，其余情况下它被忽略。

具体说明：sem\_open(const char\* name, unsigned int value)使用该函数完成对信号量的创建，为了完成选手之间以及选手与裁判之间的顺序执行，用户 1 的信号量创建的 value 值设置为 1，将选手二以及裁判的信号量设置为 0，这样开始选手二以及裁判会因为 value 值为 0 从而进入等待。

#### (6) sem\_wait()

原型：int sem\_wait(sem\_t \*sem)

功能：信号量的 P 原子操作（检查信号量是不是为负值，如果是，则停下来。

返回值：返回 0 表示成功，返回-1 表示失败。

参数说明：\*sem, 信号量指针，函数内调用参数表示调用某信号量。

具体说明：对于进程是否需要等待的判断，不能用简单的 if 语句，而应该用 while() 语句，假设现在 sem=-1，生产者往缓冲区写入了一个数，sem=0<=0，此时应该将等待队列队首的进程唤醒。当被唤醒的队首进程再次调度执行，从函数 wait\_task 退出，不会再执行 if 判断，而直接从 if 语句退出，继续向下执行。而等待队列后面被唤醒的进程随后也会被调度执行，同样也不会执行 if 判断，退出 if 语句，继续向下执行，这显然是不应该的。因为生产者只往缓冲区写入了一个数，被等待队列的队首进程取走了，由于等待队列的队首进程已经取走了那个数，它应该已经将 sem 修改为 sem=-1，其他等待的进程应该再次执行 if 判断，由于 sem=-1<0，会继续睡眠。

#### (7) sem\_post()



原型: `int sem_post(sem_t *sem)`

功能: 信号量的 V 原子操作 (检查信号量的值是不是为 0, 如果是, 表示有进程在睡眠等待, 则唤醒队首进程, 如果不是, 向下执行)。

返回值: 返回 0 表示成功, 返回-1 表示失败。

参数说明: \*sem, 信号量指针, 函数内调用参数表示调用某信号量。

具体说明: 调用该函数, 实现 value 值+1, 在本程序中选手一向管道输入完成后调用该函数, 将选手二的信号量从等待队列中移出, 完成选手二的激活, 同时选手二向管道输入完成之后调用函数释放裁判进程的信号量, 从而激活裁判进程读取管道数据。

#### (8) `sem_unlink`

原型: `int sem_unlink(const char *name)`

功能: 删除名为 name 的信号量。

返回值: 返回 0 表示成功, 返回-1 表示失败。

参数说明: \*sem, 信号量指针, 函数内调用参数表示调用某信号量。

具体说明: 在程序的最后完成结果的输出后将创建的三个信号量, `signal_1`, `signal_2`, `signal_jud` 删除回收。

## 3 算法及数据结构设计

### 3.1 数据结构设计

#### 3.1.1 管道通讯

管道的数据结构是一种特殊文件的数据结构。一个管道实际上就是个只存在于内存中的文件，对这个文件的操作要通过两个已经打开文件进行，它们分别代表管道的两端。管道是一种特殊的文件，它不属于某一种文件系统，而是一种独立的文件系统，有其自己的数据结构。

本次课设我们用 fd1[]、fd2[] 来作为输选手出招的管道，用 fd3[]、fd4[] 来作为传输选手出招总数的管道。同时也设置了整型变量 len 和 tmp 来读取管道返回出招总数的返回值以及判定函数的返回值。

另外又置两个字符数组 buf1, buf2, data1, data2, 分别用来存放选手 2 的出拳，选手 1 出拳，选手 1 的出拳总情况，选手 2 的出拳总情况。设置整型变量 p1, p2, ping, 用来统计选手 1 的胜局和选手 2 的胜局，以及平局的数量。

表 3.1 数据结构定义表

```
int fd1[2], fd2[2];/*传输选手出招的管道*/
int fd3[2], fd4[2];/*传输选手出招总数的管道*/
char buf1[1000], buf2[1000], str2[1000];/*buf: 暂时存放选手出招（字符串形式）的字符数组*/
char data1[1000], data2[1000];/*data: 暂时存放选手出招总数（字符串形式）的字符数组*/
int len, tmp; /*len 读取管道返回出招总数的返回值*/ /*tmp 判定函数的返回值*/
int i, j;
int p1 = 0, p2 = 0, ping = 0; /*胜利局数，平局数*/
int num1_0 = 0, num1_1 = 0, num1_2 = 0; /*管道所储存的选手出招所定义的整形变量*/
int num2_0 = 0, num2_1 = 0, num2_2 = 0; /*选手储存出招数所定义的整形变量*/
int n1_0 = 0, n1_1 = 0, n1_2 = 0;
int n2_0 = 0, n2_1 = 0, n2_2 = 0;
int num = 0, flag = 0; /*num 存放选手出招总数，flag 表示选手是否出拳超时的标识*/
```

下图为本程序所用四个管道的关系图：

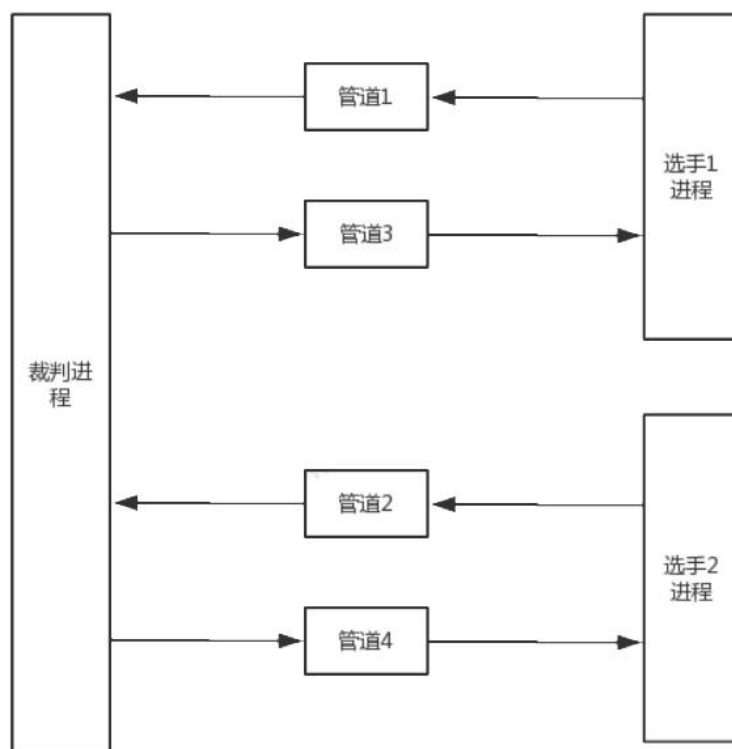


图 3.1 系统所用四个管道的关系

### 3.1.2 信号量

信号量是描述资源可用性的计数器，信号量可以通过创建一个值为 1 的信号量来专门锁定某个对象，如果信号量的值大于零，则资源可用，进程分配“资源的一个单元”，信号量减少一个。在多个进程之间设置一个信号量 `sem_id`，当一个进程执行 `sem_wait` 时，先判断这个 `sem_id` 是否为 0，为 0 则 wait。另一个进程执行 `sem_post` 把 value 加一，这样如果有 `sem_wait` 的线程则检查到 `sem_id` 不再为 0，则返回。继续执行。这样就完成了多个进程之间的同步。在一个进程执行之前先 `sem_wait` 检查一下资源是否可用，可用则把信号量减一并继续执行，执行结束之后再 `sem_post` 一下，把信号量加一，这样可以唤醒另外一个正在等待资源的进程，不可用则等待其他进程释放资源。这样就实现了进程的同步。

首先定义 `int sem_open(sem_t *sem, int pshared, unsigned int value)`、`sem_wait(sem_t *sem)`、`sem_post(sem_t *sem)`、`sem_unlink(sem_t *sem)` 函数，其中：

(1) `int sem_open(sem_t *sem, int pshared, unsigned int value)`: sem

为指向信号量结构的一个指针；pshared 不为 0 时此信号量在进程间共享，否则只能为当前进程的所有线程共享；value 给出了信号量的初始值。

表 3.2 信号量创建表

```
sem_t *signal_1, *signal_2, *mutex, *signal_jud;  
signal_1 = (sem_t*)sem_open("signal_1", 1);  
signal_2 = (sem_t*)sem_open("signal_2", 0);  
signal_jud = (sem_t*)sem_open("signal_jud", 0);
```

(2) `sem_wait( sem_t *sem )`: `sem_wait` 是一个原子操作，它的作用是从信号量的值减去一个“1”，但它永远会先等待该信号量为一个非零值才开始做减法。如果对一个值为 0 的信号量调用 `sem_wait()`，这个函数就会原地等待直到有其它线程增加了这个值使它不再是 0 为止。

表 3.3 减少信号量操作表

```
sem_wait(signal_1);  
sem_wait(signal_2);  
sem_wait(signal_jud);
```

(3) `sem_post( sem_t *sem )`: 函数 `sem_post( sem_t *sem )` 用来增加信号量的值。当有线程阻塞在这个信号量上时，调用这个函数会使其中的一个线程不在阻塞，选择机制同样是由线程的调度策略决定的。

表 3.4 增加信号量操作表

```
sem_post(signal_2);  
sem_post(signal_jud);  
sem_post(signal_1);
```

(4) `sem_unlink( sem_t *sem )`: 用来释放信号量 `sem`。

### 3.1.3 读写文件的缓冲器

文件是指存储在外部存储介质上的、由文件名标识的一组相关信息的集合。由于 CPU 与 I/O 设备间速度不匹配。为了缓和 CPU 与 I/O 设备之间速度不匹配矛盾。文件缓冲区是用以暂时存放读写期间的文件数据而在内存区预留的一定空间。使用文件缓冲区可减少读取硬盘的次数。

(1) `open` 函数: `open` 函数创建文件 `txt`，如果文件已存在，则清除文件内容。

```
fd = open("data.txt", O_WRONLY | O_TRUNC | O_CREAT);
```

(2) write 函数：将结果写入文档。

```
write(fd2[1], buf1, BUF);  
write(fd1[1], buf2, BUF);
```

## 3.2 算法设计

### 3.2.1 程序整体算法设计

(1) 程序整体的设计，首先完成系统所需相关变量的定义并添加支持信号量的系统调用函数，接着创建选手 1、选手 2 和裁判的信号量。

(2) 接着以局数为轮数完成相关的比赛流程，依次创建三个进程。由于进程之间执行是无序的，因此可以用信号量机制控制裁判和两个选手三个进程之间的并发执行顺序，使裁判进程在两个选手进程执行一轮以后裁定他们之间的胜负。

(3) 同时选手在出拳时可以根据对方之前回合出拳的汇总信息来调整自己的出拳，根据出拳算法出拳，直到比赛轮数结束为止，结束后输出总的结果，总流程如下所示：

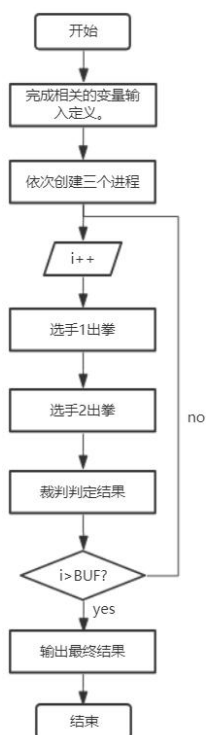


图 3.7 程序整体流程图

### 3.2.2 选手 1 出拳算法设计

选手 1 先出招，调用 `fork()` 函数创建选手 1 进程，随后激活进程，首先设置一个随机延时函数表示选手的出拳时间，超过一定时间直接判输，调用 `srand(time(NULL))` 函数从该时刻开始输入随机数 `rand() % 3`，如果选手二的历史出拳总数中，某一项出招所占比例较另外两招数量多，那么选手 1 的出招就会随之改变为相对的克制招数，随后将数据写入管道，完成输入后随机挂机等待，激活选手二进程。选手 1 的出拳预测如下图所示。

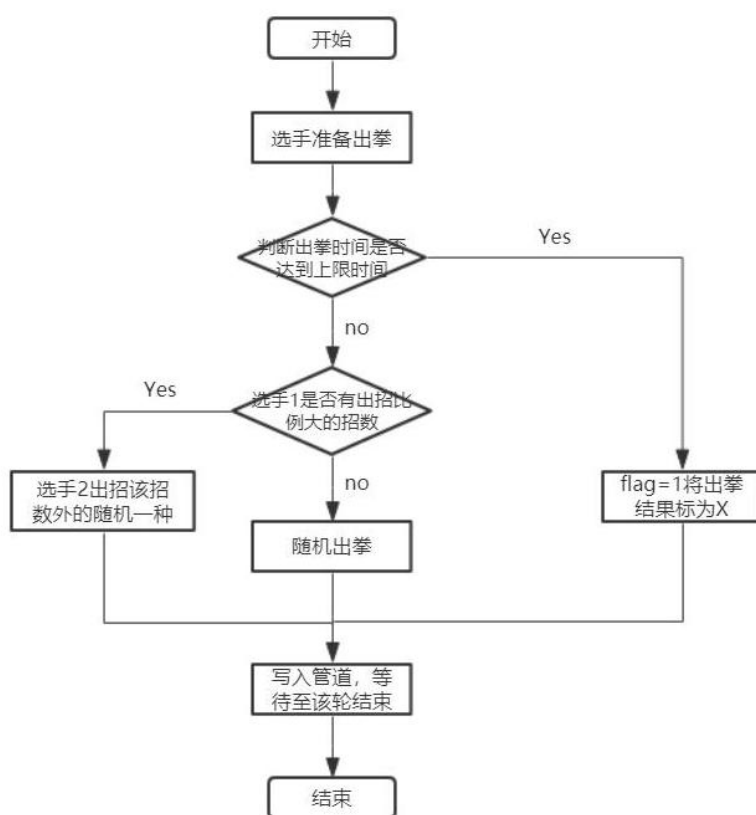


图 3.8 选手 1 出拳流程图

### 3.2.3 选手 2 出拳算法设计

选手 2 随即出招，调用 `fork()` 函数创建选手 2 进程，随后激活进程，首先设置一个随机延时函数表示选手的出拳时间，超过一定时间直接判输，调用 `srand(time(NULL))` 函数从该时刻开始输入随机数 `rand() % 3`，如果选手 1 的历史出拳总数中，某一项出招所占比例较另外两招数量多，那么选手 2 的出招就会

随之改变，较选手 1 而言，选手 2 更为保守，选手 2 将会从剩下的两个招式随机出招一个，这样的算法即便选手 1 没有出比例较大的招式，选手 2 仍有机会与选手 1 打平手或者获胜，理想状态下，选手 1 若一直出同一个招式，选手 2 的胜率将会达到  $3/4$ ，随后将数据写入管道，完成输入后随机挂机等待。选手 2 的出拳预测如下图所示。

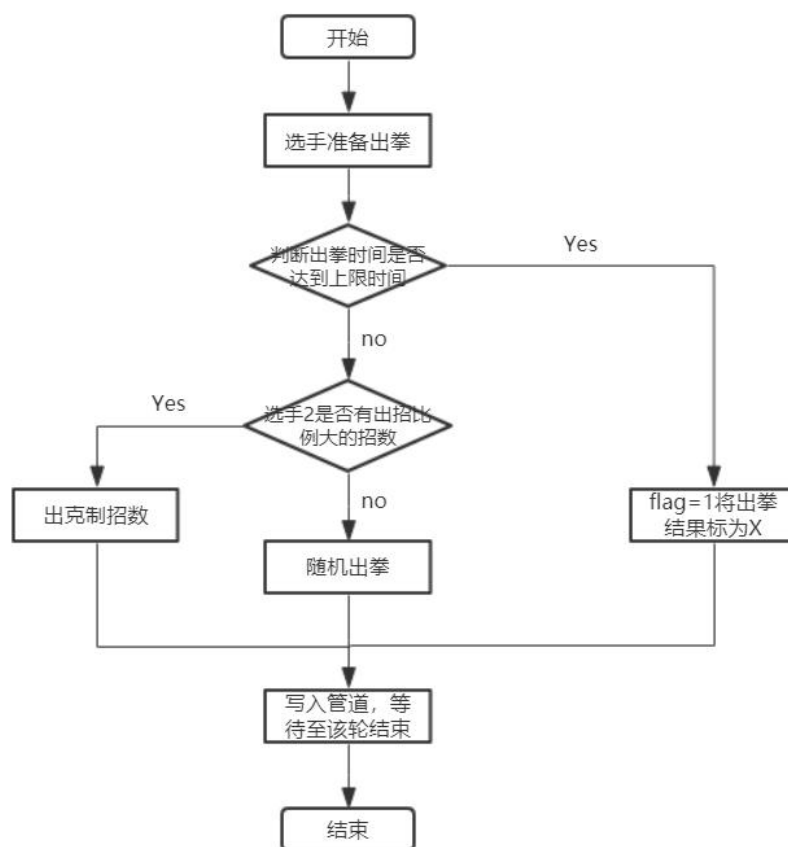


图 3.9 选手 2 出拳流程图

### 3.2.4 裁判判定设计

首先在该轮中，等待选手 1，2 完成出拳后，向管道 1，2 分别读取出拳结果，首先判断是否存在出拳超时的选手，记录相关的超时局数，如果没有则根据选手出拳结果按照 `judge` 函数所规定的规则判断比赛结果。然后将双方的出拳结果告知选手，在所有对局完成后结束比赛，宣布最后的比赛结果裁判判定流程如下所示。

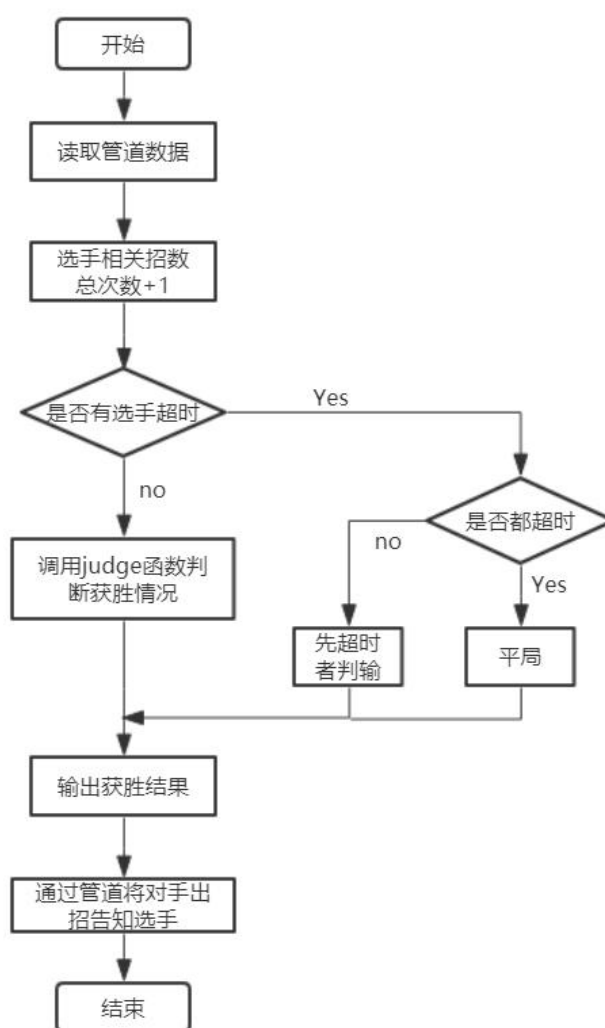


图 3.10 裁判裁定流程图

### 3.2.5 出拳时间算法设计

针对双方选手，出拳的时间可能不相同，如果选手出招超时，那么该选手将会被判输，超时信息将由管道传给裁判判定超时。针对选手的出拳时间，利用随机数模拟选手的出拳时间， $j = \text{rand}() \% 3$  实现出拳时间从而后面进行判断，算法流程如下所示。



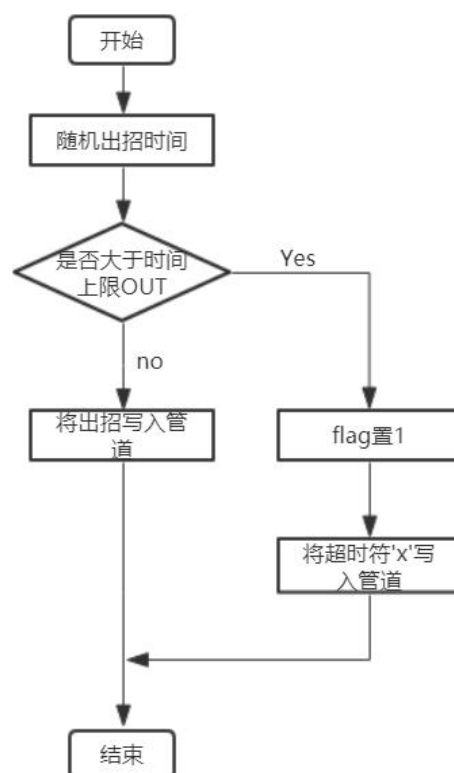


图 3.11 出拳时间算法流程图

### 3.2.6 信号量并发控制算法设计

在在程序开始时首先定义三个信号量，分别控制选手 1 进程，选手 2 进程，裁判进程，为了保证进程间的顺序执行，在调用 `(sem_t*)sem_open("signal_1", 1)` 函数时对于三个进程之间的 value 初值进行设定，进程间的调用顺序是：首先选手 1 输入该局出招招数，调用 `sem_wait()` 完成出招后由于 value 值-1 进程 1 进入等待，所以将 `signal_1` 的 value 值设置为 1，在进程 1 完成一次输入后随机调用函数 `sem_post()` 函数使得选手 2 的 value 值+1 从等待状态变为激活状态，接着开始选手 2 的出招，先调用 `sem_wait()`，在完成一次出招后随即等待，接着调用 `sem_post()` 激活裁判进程，对该局完成判定后对进程 1 调用 `sem_post()` 激活开始下一轮，如此往复，直到比赛轮数完成，控制流程如下所示。

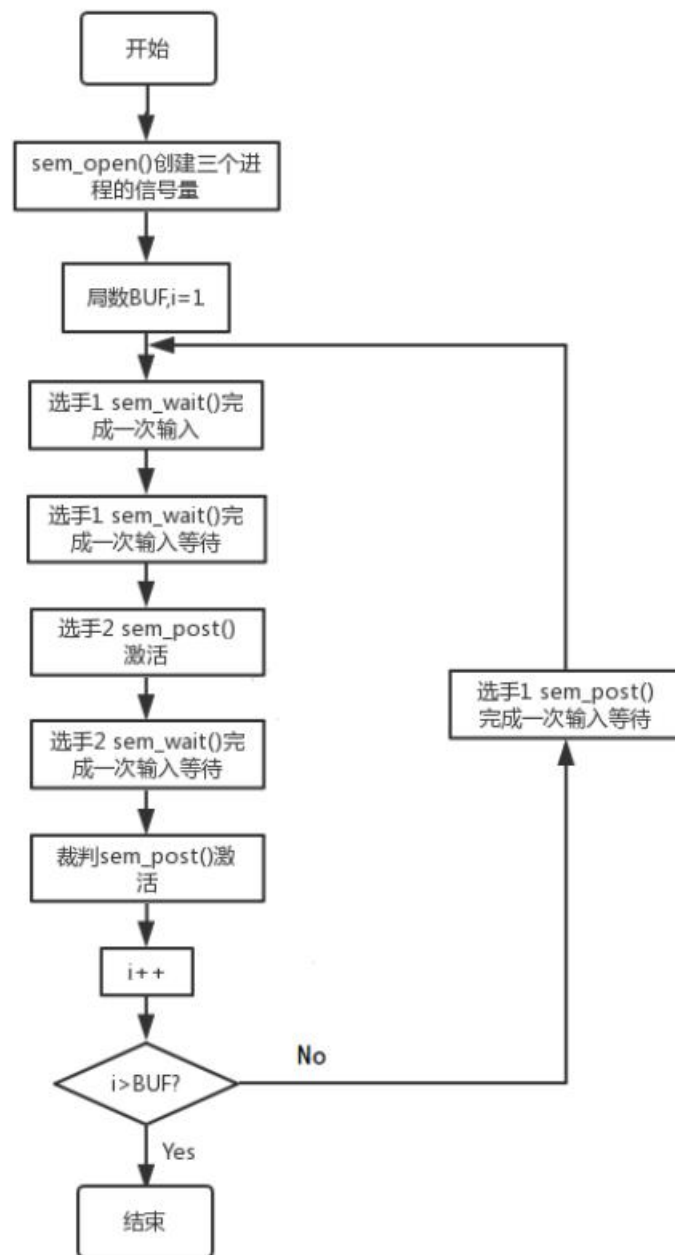


图 3.12 信号量并发控制流程图

## 4 程序测试与验证

### 4.1 测试分析

运用 Linux 应用程序导入项目，再将写好的程序存到软盘 B 中。点击调试，输入“`mcop y b:design.c design.c`”指令然后再输入“`gcc design.c -o design`”指令编译程序，最后输入“`./...`”执行程序，执行结果截图见图 4.1：

```
[/usr/root]# mcopy b:design.c design.c  
[/usr/root]# gcc design.c -o design  
[/usr/root]# ./design
```

图 4.1 编译执行 design.c

我们将测试以下五种情况：

- (1) 当回合数为-1 时，查看其输出结果。
- (2) 当回合数为 0 时，查看其输出结果。
- (3) 当回合数为 10、限制时间为 20ms 时，查看其输出结果，并查看文件保存情况。
- (4) 当回合数为 100、限制时间为 20ms 时，查看其输出结果，并查看文件保存情况。
- (5) 当回合数为 300、限制时间为 20ms 时，查看其输出结果，并查看文件保存情况。

### 4.2 测试结果

#### 4.2.1 总体程序测试

- (1) 当输入回合数-1 时：

执行应用程序，查看 Linux 操作系统的运行情况，观察输出结果，执行过程如图所示，输出请输入正确局数。

```
Please input game num and limit time:  
-1 1  
Please input right game num!
```

图 4.2 输入异常示例图

(2) 当输入回合数 0 时:

执行应用程序, 查看 Linux 操作系统的运行情况, 观察输出结果, 执行过程如图所示, 输出游戏尚未开始。

```
Please input game num and limit time:  
0 1  
The game has not started
```

图 4.3 0 回合示例图

(3) 当输入回合数为 10、限制时间为 20ms 时:

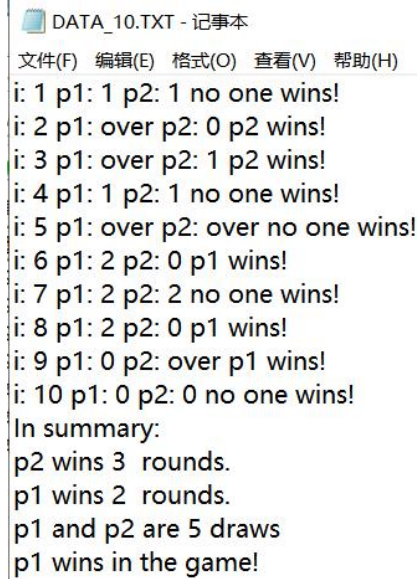
执行应用程序, 查看 Linux 操作系统的运行情况, 观察输出结果, 执行过程如图所示, 显示 p1, p2 双方的出拳情况, 并显示该回合的胜负结果。回合结束后, 显示本场比赛结果和 p1, p2 双方各赢局数, 并显示最终比赛胜负结果选手 p1 赢得比赛, 查看 data.txt 文件保存成功。

```
Please input game num and limit time:  
10 20
```

图 4.4 10 回合限时 20ms 示例图

```
i = 10, p1 pid is 14, p1:0  
i = 10, p2 pid is 15, p2:0  
i = 10, jud pid is 16 no one wins!  
  
In summary:  
p1 wins 3 rounds.  
p2 wins 2 rounds.  
p1 and p2 are 5 draws  
p1 wins in the game!  
write file successfully!
```

图 4.5 10 回合胜负结果示例图



```

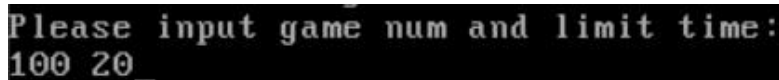
DATA_10.TXT - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
i: 1 p1: 1 p2: 1 no one wins!
i: 2 p1: over p2: 0 p2 wins!
i: 3 p1: over p2: 1 p2 wins!
i: 4 p1: 1 p2: 1 no one wins!
i: 5 p1: over p2: over no one wins!
i: 6 p1: 2 p2: 0 p1 wins!
i: 7 p1: 2 p2: 2 no one wins!
i: 8 p1: 2 p2: 0 p1 wins!
i: 9 p1: 0 p2: over p1 wins!
i: 10 p1: 0 p2: 0 no one wins!
In summary:
p2 wins 3 rounds.
p1 wins 2 rounds.
p1 and p2 are 5 draws
p1 wins in the game!

```

图 4.6 10 回合查看文本示例图

(4) 当输入回合数为 100、限制时间为 20ms 时:

执行应用程序, 查看 Linux 操作系统的运行情况, 观察输出结果, 执行过程如图所示, 显示 p1, p2 双方的出拳情况, 并显示该回合的胜负结果。回合结束后, 显示本场比赛结果和 p1, p2 双方各赢局数, 并显示最终比赛胜负结果选手 p1 赢得比赛, 查看 data.txt 文件保存成功。

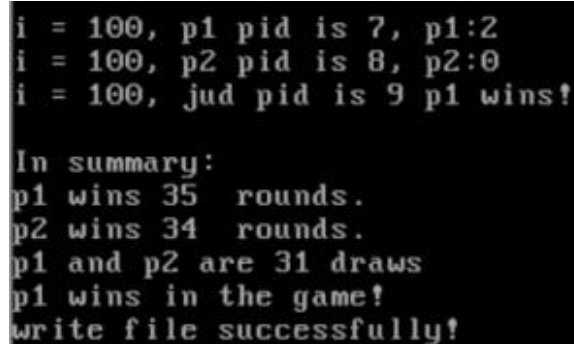


```

Please input game num and limit time:
100 20

```

图 4.7 100 回合限时 20ms 示例图



```

i = 100, p1 pid is 7, p1:2
i = 100, p2 pid is 8, p2:0
i = 100, jud pid is 9 p1 wins!

In summary:
p1 wins 35 rounds.
p2 wins 34 rounds.
p1 and p2 are 31 draws
p1 wins in the game!
write file successfully!

```

图 4.8 100 回合胜负结果示例图

```
i: 87 p1: 2 p2: 2 no one wins!
i: 88 p1: 0 p2: over p1 wins!
i: 89 p1: 0 p2: 0 no one wins!
i: 90 p1: over p2: 1 p2 wins!
i: 91 p1: 1 p2: over p1 wins!
i: 92 p1: 0 p2: over p1 wins!
i: 93 p1: 1 p2: 1 no one wins!
i: 94 p1: 0 p2: over p1 wins!
i: 95 p1: over p2: over no one wins!
i: 96 p1: 2 p2: over p1 wins!
i: 97 p1: 1 p2: 2 p1 wins!
i: 98 p1: over p2: over no one wins!
i: 99 p1: 1 p2: over p1 wins!
i: 100 p1: 2 p2: 0 p1 wins!
In summary:
p2 wins 35 rounds.
p1 wins 34 rounds.
p1 and p2 are 31 draws
p1 wins in the game!
```

图 4.9 100 回合查看文本示例图

(5) 当输入回合数为 300、限制时间为 20ms 时:

执行应用程序，查看 Linux 操作系统的运行情况，观察输出结果，执行过程如图所示，显示 p1, p2 双方的出拳情况，并显示该回合的胜负结果。回合结束后，显示本场比赛结果和 p1, p2 双方各赢局数，并显示最终比赛胜负结果选手 p1 赢得比赛，查看 data.txt 文件保存成功。

```
Please input game num and limit time:
300 20
```

图 4.10 300 回合限时 20ms 示例图

```
i = 300, p1 pid is 19, p1:0
i = 300, p2 pid is 20, p2:0
i = 300, jud pid is 21 no one wins!

In summary:
p1 wins 112 rounds.
p2 wins 109 rounds.
p1 and p2 are 79 draws
p1 wins in the game!
write file successfully!
[/usr/root]#
```

图 4.11 300 回合胜负结果示例图

```
i: 283 p1: 0 p2: over p1 wins!
i: 284 p1: over p2: 1 p2 wins!
i: 285 p1: 0 p2: 2 p2 wins!
i: 286 p1: 2 p2: 0 p1 wins!
i: 287 p1: over p2: over no one wins!
i: 288 p1: 0 p2: over p1 wins!
i: 289 p1: 2 p2: over p1 wins!
i: 290 p1: 0 p2: 0 no one wins!
i: 291 p1: over p2: 2 p2 wins!
i: 292 p1: 0 p2: 1 p1 wins!
i: 293 p1: 1 p2: over p1 wins!
i: 294 p1: over p2: 2 p2 wins!
i: 295 p1: 2 p2: over p1 wins!
i: 296 p1: 1 p2: 0 p2 wins!
i: 297 p1: 0 p2: 1 p1 wins!
i: 298 p1: 0 p2: 2 p2 wins!
i: 299 p1: 1 p2: 2 p1 wins!
i: 300 p1: 0 p2: 0 no one wins!
In summary:
p2 wins 112 rounds.
p1 wins 109 rounds.
p1 and p2 are 79 draws
p1 wins in the game!
```

图 4.12 Window 下查看文本

### 4.2.1 选手出拳算法测试

(1) 选手 1 出拳算法测试：修改选手 2 出拳算法为随机出拳，进而测试选手 1 的出拳算法，程序文件为 a1.c。

```
[/usr/root]# ./a1
Please input game num and limit time:
100 20
```

图 4.13 运行选手 1 算法

① 回合数为 100，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:
p1 wins 26 rounds.
p2 wins 34 rounds.
p1 and p2 are 40 draws
p2 wins in the game!
write file successfully!
[/usr/root]#
```

图 4.14 100 回合运行效果

② 回合数为 500，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:
p1 wins 176 rounds.
p2 wins 165 rounds.
p1 and p2 are 159 draws
p1 wins in the game!
write file successfully!
[/usr/root]#
```

图 4.15 500 回合运行效果

- ③ 回合数为 1000，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:
p1 wins 350 rounds.
p2 wins 318 rounds.
p1 and p2 are 332 draws
p1 wins in the game!
write file successfully!
[/usr/root]#
```

图 4.16 1000 回合运行效果

可以看出选手 1 的出拳算法在回合数少的时候胜局少于败局，胜率偏低，而在回合数多的时候胜率会大大提高。

- (2) 选手 2 出拳算法测试：修改选手 1 出拳算法为随机出拳，进而测试选手 2 的出拳算法，程序文件为 a2.c。

```
[/usr/root]# ./a2
Please input game num and limit time:
100 20
```

图 4.17 运行选手 2 算法

- ① 回合数为 100，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:
p1 wins 26 rounds.
p2 wins 34 rounds.
p1 and p2 are 40 draws
p2 wins in the game!
write file successfully!
[/usr/root]#
```

图 4.18 100 回合运行效果

- ② 回合数为 500，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:
p1 wins 176 rounds.
p2 wins 165 rounds.
p1 and p2 are 159 draws
p1 wins in the game!
write file successfully!
[/usr/root]#
```



图 4.19 500 回合运行效果

③ 回合数为 1000，出拳限制时间为 20ms，运行效果如图所示：

```
In summary:  
p1 wins 350 rounds.  
p2 wins 318 rounds.  
p1 and p2 are 332 draws  
p1 wins in the game!  
write file successfully!  
[/usr/root]#
```

图 4.20 1000 回合运行效果

可以看出选手 2 的出拳算法在回合数多的时候胜局少于败局，胜率偏低，而在回合数少的时候胜率会大大提高。

（3）通过以上两种算法的测试，我们可以总结出来选手 1 算法在回合数多的时候胜率高；相反选手 2 算法在回合数少的时候胜率高。因此，选手如果想要提高胜率，所用的出拳算法可以根据回合数的多少来选择对应胜率高的算法。

## 5 结论

本课设主要是通过管道来实现进程间的通信，进而完成石头、剪刀、布的游戏。在程序中我们一共创建了三个进程，其中一个进程为裁判进程，另外两个为选手进程。选手1预测是根据上一局选手2出拳时在剪刀、石头、布这三者出了哪一个来判断选手2接下来的出拳结果。而选手2预测是根据选手1出拳中剪刀、石头、布这三者哪一者次数最多来预测选手1接下来的出拳结果。而裁判负责的是将选手的胜负情况以及输赢情况及时的反馈给选手，以便于他们的预测和出拳。并且我们设计了如果程序中如果出现选手出拳超时，则直接判定它输的这一算法。三个进程之间相互独立，信息的传输通过四个管道来实现。最后将胜负的情况写入文件。

最后，通过不同回合数程序的测试，得到最终的测试结果。如果在 0-100 回合书中任意选择一个回合数进行输入时，我们得到的结论就是：选手 2 获胜概率比较大，但是随着回合数的增大，选手 1 获胜的概率也就越大，平局的概率也跟着增大。由于大部分测试回合数都比较大，所以选手 1 预测算法更优。如果在回合数较大中选择固定一种，进行多次输入，我们得到的结论就是：无论输入多少次相同的回合数，最终结果都是一样的，因为我们设计的选手 1 和选手 2 的预测算法是固定的，那么输入一样的回合数，预测的结果也是相同的，不会出现相差概率特别大的情况。

## 参考文献

- [1]北京英真时代. Linux 内核实验教程\_2.1. 北京英真时代科技有限公司编写
- [2]赵炯. Linux 内核完全注释. 北京:机械工业出版社, 2004
- [3]赵炯. Linux 内核完全剖析——基于 0.12 内核. 北京: 机械工业出版社, 2008
- [4]拉芙(RobertLove).Linux 内核设计与实现(第3版). 北京:机械工业出版社, 2011
- [5]汤小丹等.计算机操作系统（第3版）.西安电子科技大学出版社, 2007.05

## 附录 程序清单

```

#define __LIBRARY__
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<errno.h>
#include<stdlib.h>
#include<string.h>

int judge(char a, char b);
void usleep(int num);

typedef void sem_t;
#define __NR_sem_open 87
#define __NR_sem_wait 88
#define __NR_sem_post 89
#define __NR_sem_unlink 90
_syscall2(int, sem_open, const char*, name, unsigned int, value)
_syscall1(int, sem_wait, sem_t *, sem)
_syscall1(int, sem_post, sem_t *, sem)
_syscall1(int, sem_unlink, const char *, name)

/*限制的出拳时间*/
#define OUT 3

/*0:石头 1:剪刀 2:布 X:超时*/

int main()
{
    int BUF;
    int fd=0;
    int fd1[2], fd2[2];/*传输选手出招的管道*/
    int fd3[2], fd4[2];/*传输选手出招总数的管道*/
    char buf1[1000], buf2[1000],str2[1000];/*buf: 暂时存放选手出招（字符串形式）的字符数组*/
    char data1[1000], data2[1000]; /*data: 暂时存放选手出招总数（字符串形式）的字符数组*/

```

```
pid_t pid1, pid2, jud; /*进程*/
int len, tmp; /*len 读取管道返回出招总数的返回值**tmp 判定函数的返回值*/
int i, j;
int p1 = 0, p2 = 0, ping = 0; /*胜利局数，平局数*/
    /*管道所储存的选手出招所定义的整形变量*/
int num1_0 = 0, num1_1 = 0, num1_2 = 0;
int num2_0 = 0, num2_1 = 0, num2_2 = 0;
    /*选手储存出招数所定义的整形变量*/
int n1_0 = 0, n1_1 = 0, n1_2 = 0;
int n2_0 = 0, n2_1 = 0, n2_2 = 0;
int num = 0, flag = 0; /*num 存放选手出招总数，flag 表示选手是否出拳超时的标识*/

/*定义初始化信号量*/
sem_t *signal_1, *signal_2, *mutex, *signal_jud;
signal_1 = (sem_t*)sem_open("signal_1", 1);
signal_2 = (sem_t*)sem_open("signal_2", 0);
signal_jud = (sem_t*)sem_open("signal_jud", 0);

printf("Please input game num: ");
printf("\n");
scanf("%d", &BUF);

/*创建管道*/
if (pipe(fd1) < 0)
{
    perror("failed to pipe");
    exit(1);
}
if (pipe(fd2) < 0)
{
    perror("failed to pipe");
    exit(1);
}
if (pipe(fd3) < 0)
{
    perror("failed to pipe");
    exit(1);
}
if (pipe(fd4) < 0)
{
    perror("failed to pipe");
    exit(1);
}
```

```

/*创建选手 1 进程*/
pid1 = fork();
if (pid1 == 0)
{
    printf("create p1 pid successfully\n\n");
    /*从当前时间段取随机数*/
    srand(time(NULL));
    for (i = 1; i <= BUF; i++)
    {
        /*进程被堵塞，等待激活*/
        sem_wait(signal_1);
        /*随机设置延时时长，作为选手的出牌时间*/
        j = rand() % 3;
        /*sleep(j);*/
        /*判断是否超时*/
        if (j >= OUT) {
            flag = 1;
            sprintf(buf2, "%c", 'X');
        }

        /*不超时的情况下*/
        if (i > 1 && !flag) {
            len = read(fd3[0], data1, BUF + 10);/*读取管道 3 从裁判进程传输的选手 2
            的出过的石头剪刀布的总次数*/
            /*printf("data1: %s\n", data1);*/
            sscanf(data1, "%d %d %d", &n2_0, &n2_1, &n2_2);/*将 data1 存储的字符串转
            化为整数存储，data1 数据分割以空格分开*/
            num = n2_0 + n2_1 + n2_2; /*统计选手 2 的总对局数*/
            /*判断是否选手 2
            存在出招比例较大的招数，存在则出相克制的招数*/
            if (n2_0 > num*3*1.0/8) sprintf(buf2, "%d", 2);
            else if (n2_1 > num*3*1.0/8) sprintf(buf2, "%d", 0);
            else if (n2_2 > num*3*1.0/8) sprintf(buf2, "%d", 1);
            /*否则按照随机数
            输入*/
            else sprintf(buf2, "%d", rand() % 3);
        }
        else if (i == 1) sprintf(buf2, "%d", rand() % 3);/*如果是第一局，选手 2 还未出招，
        无法计算比例，直接输入随机数*/
        printf("i = %d, p1 pid is %d, p1:%c\n", i, getpid(), buf2[0]);/*输出当前的进程号和
        局数，证明信号量控制是按照顺序执行的*/
        write(fd2[1], buf2, BUF);/*将出招结果写入管道*/
        usleep(50);
        /*激活进程 2*/
    }
}

```

```

        sem_post(signal_2);
    }
    exit(0);
}
/*创建选手 2 进程*/
pid2 = fork();
if (pid2 == 0)
{
    printf("create p2 pid successfully\n");
    srand(time(NULL) + BUF);
    for (i=1; i <= BUF; i++)
    {
        /*进程被堵塞，等待激活*/
        sem_wait(signal_2);
        /*随机设置延时时长，作为选手的出牌时间*/
        j = rand() % 3;
        /*sleep(j);*/
        /*判断是否超时*/
        if (j >= OUT) {
            flag = 1;
            sprintf(buf1, "%c", 'X');
        }
        if (i > 1 && !flag) {
            len = read(fd4[0], data2, BUF + 10);/*读取管道 4 从裁判进程传输的选手 1
的出过的石头剪刀布的总次数*/
            /*printf("data2: %s\n", data2);*/
            sscanf(data2, "%d %d %d", &n1_0, &n1_1, &n1_2);/*将 data2 存储的字符串
转化为整数存储，data2 数据分割以空格分开*/
            num = n1_0 + n1_1 + n1_2; /*统计选手 2 的总对局数*/
            /*判断是否选手 1
存在出招比例较大的招数，存在则出相克制的招数*/
            if (n1_0 > num * 3 * 1.0 / 8) sprintf(buf1, "%d", 2);
            else if (n1_1 > num * 3 * 1.0 / 8) sprintf(buf1, "%d", 0);
            else if (n1_2 > num * 3 * 1.0 / 8) sprintf(buf1, "%d", 1);
            /*否则按照随机
数输入*/
            else sprintf(buf1, "%d", rand() % 3);
        }
        else if (i == 1) sprintf(buf1, "%d", rand() % 3); /*如果是第一局，选手 1 还未出招，
无法计算比例，直接输入随机数*/
        printf("i = %d, p2 pid is %d, p2:%c\n", i, getpid(), buf1[0]); /*输出当前的进程号和
局数，证明信号量控制是按照顺序执行的*/
        write(fd1[1], buf1, BUF); /*将出招结果写入管道*/
    }
}

```

```

        usleep(50);
        /*激活裁判进程*/
        sem_post(signal_jud);
    }
    exit(0);
}
/*创建裁判进程*/
jud = fork();
if (jud == 0)
{
    printf("create jud pid successfully\n");
    fd = open("data.txt", O_WRONLY | O_TRUNC | O_CREAT);/*创建文本文件记录数据
*/
    /*清空存放招式的数组，因为裁判进程需要使用*/
    memset(buf1, 0, sizeof(buf1));
    memset(buf2, 0, sizeof(buf2));
    for (i = 1; i <= BUF; i++)
    {
        /*进程被堵塞，等待激活*/
        sem_wait(signal_jud);
        printf("i = %d, jud pid is %d ", i, getpid());/*输出裁判进程号，以及局数，验证信
号量是按顺序执行的*/
        len = read(fd1[0], buf1, BUF);/*读取管道 1 存放的选手招式写入 buf1*/
        usleep(50);
        len = read(fd2[0], buf2, BUF);/*读取管道 2 存放的选手招式写入 buf2*/
        usleep(50);
        tmp = judge(buf2[0], buf1[0]);/*按照判断函数，判断选手 1，2 之间的胜者*/
        /*将所出的招式加在出招总数里面*/
        if (buf1[0] == '0') num2_0++;
        else if (buf1[0] == '1') num2_1++;
        else if (buf1[0] == '2') num2_2;
        if (buf2[0] == '0') num1_0++;
        else if (buf2[0] == '1') num1_1++;
        else if (buf2[0] == '2') num1_2++;
        /*判断是否有选手出招超时*/
        if (buf2[0] == 'X' && buf1[0] != 'X') {
            printf("p1 is time out,so p2 wins!\n\n");
            sprintf(str2, "i: %d p1: %s p2: %c p2 wins!\n", i, "over", buf1[0]);/*输出该轮
比试结果*/
            write(fd, str2, strlen(str2));
            p2++;/*胜利局数+1*/

            sprintf(data1, "%d %d %d", num2_0, num2_1, num2_2);/*将更新后的出招总

```



```

数以字符串形式存入 data1 中*/
    sprintf(data2, "%d %d %d", num1_0, num1_1, num1_2);/*将更新后的出招总
数以字符串形式存入 data2 中*/
    write(fd3[1], data1, BUF + 10);/*将字符串写入管道*/
    write(fd4[1], data2, BUF + 10);
    /*printf("num1: %s,num2: %s\n", data1, data2);*/
}
else if (buf1[0] == 'X' && buf2[0] != 'X') {
    printf("p2 is time out,so p1 wins!\n\n");
    sprintf(str2, "i: %d p1: %c p2: %s p1 wins!\n", i, buf2[0], "over");/*输出该轮
比试结果*/

    write(fd, str2, strlen(str2));
    p1++;/*胜利局数+1*/

    sprintf(data1, "%d %d %d", num2_0, num2_1, num2_2);/*将更新后的出招总
数以字符串形式存入 data1 中*/
    sprintf(data2, "%d %d %d", num1_0, num1_1, num1_2);/*将更新后的出招总
数以字符串形式存入 data2 中*/
    write(fd3[1], data1, BUF + 10);/*将跟新后的出招总数（字符串）写入管道
*/
    write(fd4[1], data2, BUF + 10);/*将跟新后的出招总数（字符串）写入管道
*/

    /*printf("num1: %s,num2: %s\n", data1, data2);*/
}

/*都超时*/
else if (buf1[0] == 'X' && buf2[0] == 'X') {
    ping++;/*平局数+1*/
    printf("p1 and p2 are both time out,so no one wins!\n\n");
    sprintf(str2, "i: %d p1: %s p2: %s no one wins!\n", i, "over", "over");
    write(fd, str2, strlen(str2));

    sprintf(data1, "%d %d %d", num2_0, num2_1, num2_2);/*将更新后的出招总
数以字符串形式存入 data1 中*/
    sprintf(data2, "%d %d %d", num1_0, num1_1, num1_2);/*将更新后的出招总
数以字符串形式存入 data2 中*/
    write(fd3[1], data1, BUF + 10);/*将跟新后的出招总数（字符串）写入管道
*/
    write(fd4[1], data2, BUF + 10);/*将跟新后的出招总数（字符串）写入管道
*/

    /*printf("num1: %s,num2: %s\n", data1, data2);*/
}
else
{

```

```

        sprintf(data1,"%d %d %d",num2_0,num2_1,num2_2);/*将更新后的出招总数
以字符串形式存入 data1 中*/
        sprintf(data2,"%d %d %d",num1_0,num1_1,num1_2);/*将更新后的出招总数
以字符串形式存入 data2 中*/
        write(fd3[1], data1, BUF+10);/*将跟新后的出招总数（字符串）写入管道*/
        write(fd4[1], data2, BUF + 10);/*将跟新后的出招总数（字符串）写入管道
*/

        /*printf("num1: %s,num2: %s\n",data1,data2);*/

                                                /*正常出招为平局
*/

        if (tmp == 0) {
            ping++;
            printf("no one wins!\n\n");
            sprintf(str2, "i: %d p1: %c p2: %c no one wins!\n", i, buf1[0], buf2[0]);/*
写入文本文档*/

            write(fd, str2, strlen(str2));/*写入文本文档*/
        }
        else {
            if (tmp > 0)/*选手 1 获胜*/
            {
                p1++;
                printf("p1 wins!\n\n");
                sprintf(str2, "i: %d p1: %c p2: %c p1 wins!\n", i, buf2[0], buf1[0]);/*
写入文本文档*/

                write(fd, str2, strlen(str2));/*写入文本文档*/
            }
            else/*选手 2 获胜*/
            {
                p2++;
                printf("p2 wins!\n\n");
                sprintf(str2, "i: %d p1: %c p2: %c p2 wins!\n", i, buf2[0], buf1[0]);/*
写入文本文档*/

                write(fd, str2, strlen(str2));/*写入文本文档*/
            }
        }
    }
    /*激活进程 1*/
    sem_post(signal_1);
}

/* 输出对局总情况*/
sprintf(str2, "In summary:\np2 wins %d rounds.\np1 wins %d rounds.\np1 and p2
are %d draws\n", p1, p2,ping);
write(fd, str2, strlen(str2));

```

```

printf("In summary:\n");
printf("p1 wins %d rounds.\n", p1);
printf("p2 wins %d rounds.\n", p2);
printf("p1 and p2 are %d draws\n",ping);
if (p1 == p2)
{
    printf("p1 and p2 rounds are same,so p1 and p2 are draws!\n");
    strcpy(str2, "p1 and p2 rounds are same,so p1 and p2 are draws!\n");
    write(fd, str2, strlen(str2));
}
else
{
    printf("%s wins in the game!\n", (p1 > p2) ? "p1" : "p2");
    sprintf(str2,"%s wins in the game!\n", (p1 > p2) ? "p1" : "p2");
    write(fd, str2, strlen(str2));
}
printf("write file successfully!\n");
}

/*等待三个进程的结束*/
waitpid(pid1, NULL, 0);
waitpid(pid2, NULL, 0);
waitpid(jud, NULL, 0);
exit(0);
}

void usleep(int num)
{
    int i;
    for (i = 1; i <= num; i++);
}

int judge(char a, char b)
{
    int r = 0;
    if (a == '3') return -1;
    else if (b == '3') return 1;
    if (a == b)
        r = 0;
    else
    {
        if (a == '0' && b == '1') r = 1;
        if (a == '0' && b == '2') r = -1;
        if (a == '1' && b == '2') r = 1;
    }
}

```

```
    if (a == '1' && b == '0') r = -1;
    if (a == '2' && b == '0') r = 1;
    if (a == '2' && b == '1') r = -1;
}
return r;
}
```

### 课程设计总结：

首先感谢董老师对本次课设的辛勤指导和检查,通过此次课程设计,使我更加扎实的掌握了有关 Linux 在进程利用管道进行通信和用信号量控制进程之间的并发方面的知识,在设计过程中虽然遇到了一些问题,但经过一次又一次的思考,一遍又一遍的检查终于找出了原因所在,也暴露出了前期我在这方面的知识欠缺和经验不足。实践出真知,通过亲自动手制作,使我们掌握的知识不再是纸上谈兵。

课程设计诚然是一门专业课,给我很多专业知识以及专业技能上的提升,同时又是一门通识课,一门辩思课,给了我许多学习的机会,给了我很多灵感,给了我莫大的空间。同时,课程设计让我感触很深,使我对抽象的理论有了具体的认识。通过这次课程设计,我掌握了管道通信的使用和测试;熟悉了程序对管道进行读写操作;了解了利用信号量控制进程并发的方法;以及如何提高进程并发速率的性能等等,掌握了 Linux 进程之间的通信和并发执行的方法和技术,通过查询资料,也了解了进程之间是如何利用管道进行通信和如何利用信号量对进程进行并发控制的原理。

课程设计过程中,也对团队精神的进行了考察,让我们在合作起来更加默契,在成功后一起体会喜悦的心情。果然是团结就是力量,只有互相之间默契融洽的配合才能换来最终完美的结果。

学生（签字）：何雨泊

日期：2020 年 7 月 8 日

**课程设计总结：**

首先感谢董老师对我本次课程设计的指导检查，通过该次设计，实现了我从课堂理论知识到实践的过程，通过团队合作分模块编写代码，我对于进程间管道通信以及信号量机制有加深性的理解。

从最开始的编写大致结构，到后面经过老师几次指导不断修改相关问题，增加细节功能等操作，认识到了怎样通过信号量控制进程之间的顺序，怎样使得游戏比赛的结果偏向个性化，怎样设计选手算法使得体现的选手思维不同，最后对比算法的优劣，在编写代码的时候稍有程序编写出错都会拖沓整个团队的进度，在经过几次的修改调试之后，在与团队沟通相关数据结构的具体使用之后，修改之后的程序整合到一起错误会大大降低，特别是相关变量的定义就有多行，必须清楚了解哪些变量是对应哪些数据结构，在查看整合之后的代码，对比较模糊的知识概念通过实践的方式有较为清晰的了解，对于管道通信以及信号量控制都有更进一步的体会，在控制延时的时候，查询了很多有关时间的函数，虽然很可惜较前的版本并没有相关库函数的定义，但是我也了解了 linux 更多函数，获益不少。

一步步感受到把理论带向实践的过程，这比浅尝辄止知识所带来的印象更为深刻，对 Linux 的使用有着更为清晰的了解。

团队合作是之前没有尝试过，对于分工和配合从最开始的分配不均，到后面分模块的分配任务，到后来的讨论，进行了很多次，但是每一次都会有新的体会，更加能感受到团队合作带来的成果，体会到了团队精神的重要性。再次感谢董老师对本次实验的指导，希望在以后的学习中能有长足的进步。

学生（签字）：



日期：2020 年 7 月 8 日

### 课程设计总结：

通过这次课程设计，我对 Linux 进程与通信方面的知识有了更加深刻的理解。刚开始接触这个课题，我们都无从下手，因为这个题目要运用的知识我还没有学过。虽然初学操作系统知识比较薄弱，但在经历了第一周的不断学习后，对管道与信号量方面的知识也有了一个初步的理解。由于前期对问题理解不是很透彻，导致了我们所做内容与老师期望的有偏差，但在老师的指导下，我们慢慢步入正轨，顺利地完成了这个课设。

回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，能够说得是苦多于甜，但是能够学到很多很多东西，同时不但能够巩固了之前实验所学的知识，还能学会管道和信号量相关的内容。通过这次课程设计使我懂得了理论与实际相结合是很重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践结合起来，从理论中得出结论，才能真正提升自己的实际动手水平和独立思考的水平。在设计的过程中遇到问题，能够说得是困难重重，但可喜的是最终都得到了解决。

在这两周的时间里我学到了很多，不仅是 Linux 的知识，还有团队合作的重要性。只有团队成员齐心协力，互相配合才能顺利完成课设。此次设计也让我明白了思路即出路，有什么不懂不明白的地方要及时请教老师或上网查询，也非常感谢老师在百忙之中抽出时间为我们指导。

学生（签字）：汪恒辉

日期：2020 年 7 月 8 日