苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 操作系统课程实践 | | | | | | | 成绩 |  |
| 指导教师 | | 王红玲 | | 同组实验者 | | 无 | | 实验日期 | 2024.3.13 | |

|  |  |
| --- | --- |
| 实 验 名 称 | 实验3 |

1. 实验目的

初步了解 Linux 系统中，创建进程和进程间通信的方法

1. 实验内容

1．实验 Linux 下创建子进程及资源共享的方法

2. 编写一个程序，用 Linux 中的 IPC 机制，完成两个进程“石头、剪子、布”的游戏。

1. 实验步骤和结果

本实验可以创建三个进程，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（＞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。

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．选择 IPC 的方法。

4．根据你所选择的 IPC 方法，创建对应的 IPC 资源。

5．完成选手进程。

6． 完成裁判进程。

－－以下要求选作：

7．决出班级的前三甲，与另外班级的前三甲比赛，决出年级冠军。

8．如果有兴趣，再把这个实验改造成网络版。即在设计时就要考虑ＩＰＣ层的封装。

1. 实验数据结构:



使用了枚举来表示剪刀石头布,在这个枚举中，SCISSORS、ROCK和PAPER分别代表剪刀、石头和布，赋予了0、1和2这三个整数值。使用这样的枚举类型而不是直接使用整数值，可以让代码更加直观，同时减少因错误使用整数值而引起的错误。

2 . 实验大小规则&存放方式：

大小规则判断后的结果同样也使用了枚举,分别表示玩家1胜利,玩家2胜利,;平局这三种情况,这个游戏中的大小规则遵循经典的石头剪刀布游戏规则：

石头（ROCK）胜剪刀（SCISSORS）

剪刀（SCISSORS）胜布（PAPER）

布（PAPER）胜石头（ROCK）

这个规则可以通过比较枚举值来实现，但直接比较枚举值不足以判断胜负，需要实现一个逻辑判断。这是因为这个游戏的胜负关系不是线性的，而是环形的。



上面的函数是用来判断剪刀石头布结果的函数,返回值使用了枚举来表示以增加代码可读性

3. 实验所选择的 IPC 方法和理由：

选择了消息队列作为进程间通信（IPC）的方法。消息队列是一种允许一个或多个进程写入和读取消息的IPC机制，这些消息存储在队列中，直到被接收进程取走。下面是选择消息队列作为IPC方法的具体理由：

理由一：同步和异步通信的灵活性

消息队列支持同步和异步的通信方式。在这个游戏中，虽然父进程需要等待子进程发送消息，说明它们的选择（石头、剪刀、布），但这个等待是非阻塞的。父进程可以继续执行其他任务，直到消息准备好被读取。这提供了一种灵活的方式来处理进程间的通信，特别是在需要处理多个子进程时。

理由二：解耦生产者和消费者

消息队列解耦了消息的生产者和消费者，使得生产者和消费者可以独立地工作，不需要同时在线。这在本程序中尤为重要，因为每个子进程（生产者）生成结果后即退出，而父进程（消费者）则需要从多个子进程收集结果。使用消息队列，可以简化这种一对多的通信模式。

理由三：易于管理和跟踪

消息队列提供了易于使用的接口来发送、接收和管理消息。它允许父进程轻松地从多个子进程接收消息，而不需要复杂的同步机制或共享内存管理。此外，每个消息都可以携带类型信息，使得父进程能够根据消息类型或其他标识符来过滤消息，这在处理不同类型的消息或来自不同源的消息时非常有用。

理由四：系统级支持和稳定性

消息队列作为操作系统提供的一种IPC机制，具有较高的稳定性和效率。它是内核级的，意味着消息传递的性能比用户空间的解决方案更优，且更能抵御进程间通信中的常见问题，如数据竞争和同步问题。

理由五：安全性

消息队列可以通过操作系统的权限模型来管理访问控制，确保只有具有适当权限的进程才能访问特定的消息队列。这对于需要考虑安全性的应用来说是一个重要特性。

1. 消息缓冲区结构



消息缓冲区结构的组成部分：

msgType：这是一个long类型的字段，用于标识消息的类型。在消息队列中，msgType用于区分不同种类的消息，使得接收进程可以根据类型选择性地接收消息。在您的程序中，所有的消息类型都被设定为GAME\_MSG\_TYPE（其值被定义为1），这是为了简化示例。在更复杂的应用中，可以使用不同的msgType值来表示不同的消息种类或来源。

round：这是一个int类型的字段，表示玩家在当前轮次中的选择。具体来说，0代表剪刀（SCISSORS），1代表石头（ROCK），2代表布（PAPER）。这个字段用于传递玩家的游戏选择，让接收方（通常是父进程）可以判断游戏的结果。

消息缓冲区结构的作用：

这个结构体允许您在单个消息中打包和传递游戏的关键信息。通过使用这种结构化的消息格式，您可以确保消息队列中的数据是自描述的，并且接收进程可以容易地解析和处理这些消息。在您的游戏程序中，父进程使用msgrcv函数根据msgType接收消息，然后根据round字段中的值来判断每个玩家的选择，并计算游戏结果。

消息队列中消息的处理：

发送消息：每个子进程通过msgsnd函数发送一个Game结构体的实例到消息队列，该实例包含了它的游戏选择。

接收消息：父进程使用msgrcv函数根据msgType从消息队列中接收消息，并根据round字段中包含的玩家选择来计算和宣布游戏结果。

5.如何创建 IPC 资源

IPC资源是通过消息队列创建的。消息队列允许一个或多个进程向另一个进程发送和接收消息，是一种重要的进程间通信（IPC）机制。程序中创建消息队列的过程如下所述：

创建消息队列

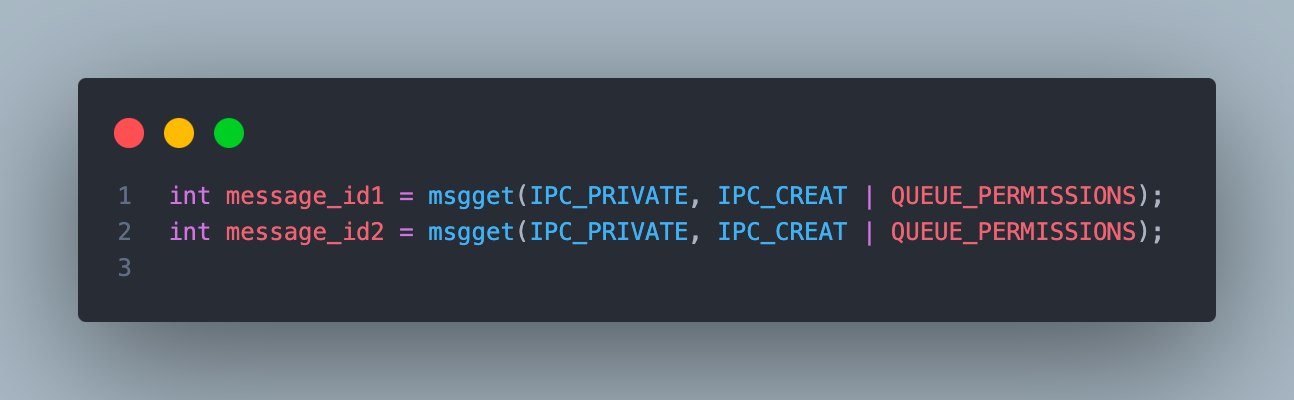
程序通过调用msgget函数创建消息队列。msgget函数的原型如下：

int msgget(key\_t key, int msgflg);

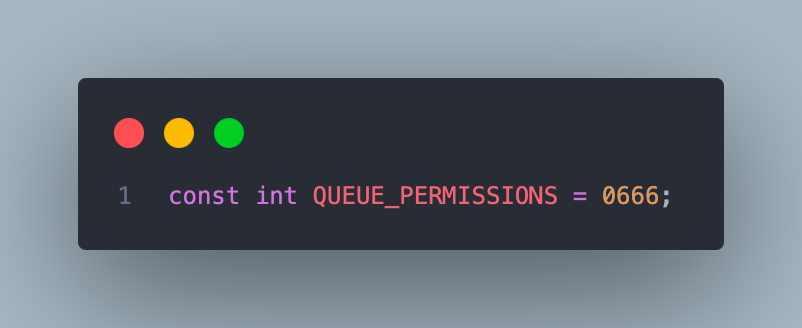
其中key 参数用于指定消息队列的访问键。在这个程序中，使用IPC\_PRIVATE作为key的值，这意味着创建一个新的、唯一的消息队列，它只能被创建它的进程及其子进程访问。

msgflg 参数用于指定消息队列的权限和创建选项。在这个程序中，使用IPC\_CREAT | QUEUE\_PERMISSIONS作为msgflg的值，其中IPC\_CREAT标志指示如果指定的key对应的消息队列不存在，则创建它。QUEUE\_PERMISSIONS通常设置为0666，提供了对消息队列的读写权限。

程序中创建IPC资源的代码：



其中,QUEUE\_PERMISSIONS在



被定义

在这段代码中，程序尝试创建两个消息队列，分别用于两个玩家的游戏结果通信。每个msgget调用都会返回一个消息队列标识符（在上述代码中分别是message\_id1和message\_id2），该标识符用于后续的消息发送和接收操作。

错误处理

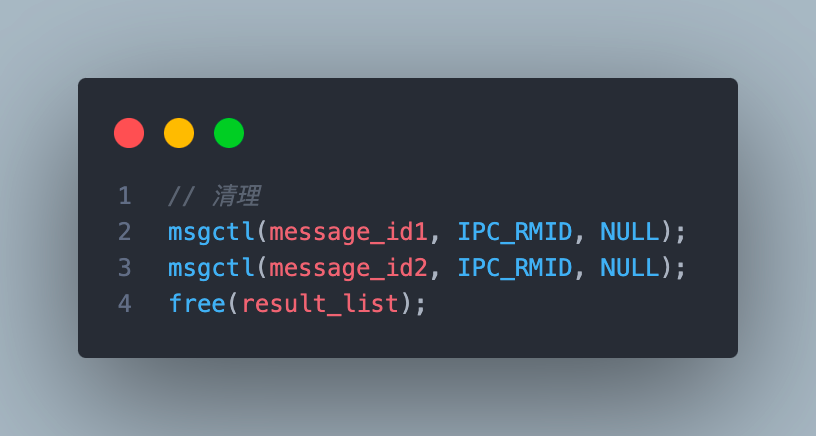
如果msgget函数调用失败（例如，因为系统资源不足或权限问题），它会返回-1。程序通过检查msgget的返回值来处理潜在的错误：



如果创建消息队列失败，程序会输出错误消息，并进行适当的资源清理和错误处理。

清理

创建的IPC资源（消息队列）在使用完毕后应当被清理，以避免资源泄露。在Linux中，可以使用msgctl函数删除消息队列：



在程序的最后，使用IPC\_RMID命令删除两个消息队列，确保资源被正确回收。

通过上述步骤，程序成功创建了所需的IPC资源（消息队列），用于实现进程间的通信。

6. 程序主要流程或关键算法：

程序的核心流程：

1. 初始化和资源准备

程序开始时，首先设置全局的随机数种子，以确保后续的随机数生成有足够的随机性。

然后，程序读取用户输入的比赛轮数，并为存放每轮结果的数组分配内存。

接着，程序通过msgget函数创建两个消息队列，分别用于两个玩家的游戏结果通信。

2. 游戏循环

程序进入一个循环，每次循环代表一轮游戏。

在每轮游戏中，程序先后为两个玩家创建子进程。

子进程：每个子进程通过调用result\_send函数生成一个随机的手势（石头、剪刀、布），并将这个手势作为消息发送到相应的消息队列。子进程完成任务后即退出。

父进程：在两个子进程都发送了游戏结果后，父进程使用msgrcv函数从两个消息队列中接收游戏结果，然后调用result\_announce函数来计算并宣布这一轮的结果（即哪个玩家赢了，或者是否平局）。

3. 结果统计和输出

对于每轮游戏的结果，父进程将其记录在先前分配的结果数组中。

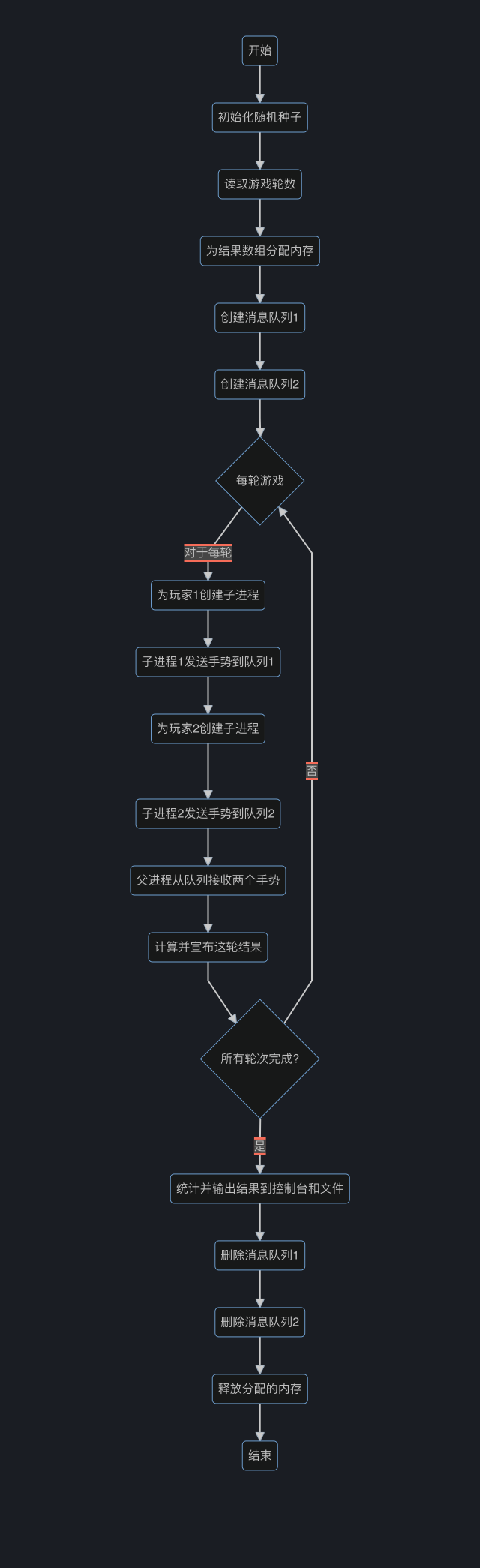
游戏结束后，父进程遍历结果数组，统计并输出每个玩家的胜利次数和平局次数。

最终结果也被写入到一个文件中，以便于后续查看。

4. 清理资源

在所有游戏轮次完成后，程序清理创建的IPC资源（消息队列），并释放分配的内存资源。

使用msgctl函数删除两个消息队列，确保不会有资源泄露。



上面的是该程序的流程图

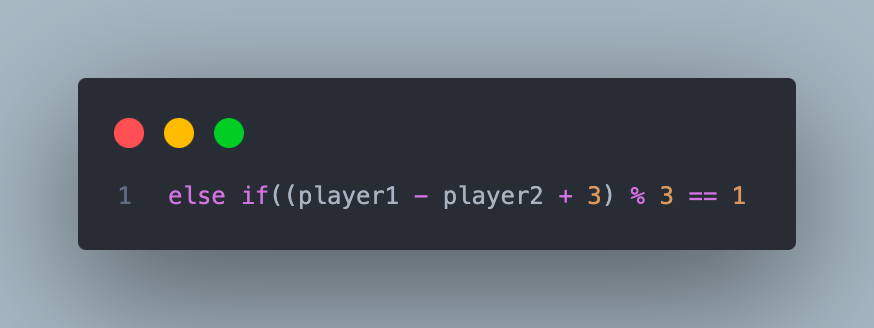
关键算法:

随机数生成：每个子进程生成一个0到2之间的随机数来代表玩家的手势（石头、剪刀、布）。这是通过调用rand函数实现的，随机数种子在每个子进程中根据当前时间和进程ID设置，以确保随机性。

胜负判断：使用result\_announce函数根据石头剪刀布的规则判断胜负，这里利用了数学关系来简化判断逻辑。具体是比较两个玩家选择的差值，通过模3运算的结果来确定胜负关系。



其中



用来快速判断玩家一剪刀,玩家二布等情况

1. 实验总结

理解IPC机制

实验深入探讨了进程间通信（IPC）的一种具体形式：消息队列。通过实践，明白了消息队列如何在独立进程间传递消息，以及它的优点，包括解耦生产者和消费者、提供同步和异步通信的灵活性，以及如何通过系统级支持实现高效和稳定的通信。

进程管理

通过创建子进程来代表不同的玩家，实验加深了对于进程创建、执行和管理的理解。特别是如何使用fork()创建子进程，以及如何通过wait()确保父进程能够正确地等待子进程完成，这对于编写并发程序来说是非常重要的基础知识。

随机数生成

在实验中，每个子进程需要生成一个随机的手势（石头、剪刀、布）。这一部分强调了如何在程序中合理使用随机数生成器，以及如何通过改变种子来确保随机性，尤其是在并发环境下。

错误处理和资源管理

实验过程中也涉及了错误处理和资源管理的重要性，比如如何在消息队列创建失败时进行错误报告和资源清理，以及在程序结束时如何释放已分配的内存和IPC资源。这些实践强调了编写健壮、可靠程序的重要性。

模块化和代码重用

通过将特定功能（如发送游戏结果、计算和宣布比赛结果等）封装成函数，实验加深了对于模块化编程和代码重用原则的理解。这种做法不仅提高了代码的可读性和可维护性，也使得功能的测试和调试更为方便。