**操作系统**

**实验报告**

**学 号\_\_\_\_\_\_\_\_\_21074121\_\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_\_何超然\_\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_\_2022.12.18\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| **报告格式(5)** | **报告内容(15)** | **教师评语** | **报告成绩** |
| **要求：项目完整、图表符合规范、没有错别字、符合模板规范。（满分5分；实际得分 ）** | **□图表内容符合计算机学科标准（满分4分；实际得分 ）**  **□可否体会设计思路实现方案/设计思路实现方案（满分10分；实际得分 ）**  **□是否提交了电子版的代码（满分10分；实际得分 ）** |  |  |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 目录

[目录 1](#_Toc153811035)

[实验二 2](#_Toc153811036)

[一、 实验目的 2](#_Toc153811037)

[二、 实验内容 2](#_Toc153811038)

[三、 实验设计 2](#_Toc153811039)

[四、 实验结果与分析 2](#_Toc153811040)

[实验三 4](#_Toc153811041)

[一、 实验目的 4](#_Toc153811042)

[二、 实验内容 4](#_Toc153811043)

[三、 实验设计 4](#_Toc153811044)

[四、 实验结果与分析 5](#_Toc153811045)

[实验四 6](#_Toc153811046)

[一、 实验目的 6](#_Toc153811047)

[二、 实验内容 6](#_Toc153811048)

[三、 实验设计 6](#_Toc153811049)

[四、 实验结果与分析 8](#_Toc153811050)

[实验五 10](#_Toc153811051)

[一、 实验目的 10](#_Toc153811052)

[二、 实验内容 10](#_Toc153811053)

[三、 实验设计 10](#_Toc153811054)

[四、 实验结果与分析 11](#_Toc153811055)

[收获与体会 12](#_Toc153811056)

[参考资料 13](#_Toc153811057)

# 实验二

1. 实验目的

学习在 Linux 操作系统中父子进程的通信方法。

1. 实验内容

编写程序实现进程的管道通信。父进程使用系统调用 pipe()创建一个无名管道，二个子

进程分别向管道各写一句话：

Child 1 is sending a message!

Child 2 is sending a message!

父进程从管道中读出二个来自子进程的信息并显示出来。

1. 实验设计

管道可以在进程间起到通信的作用，分为无名管道和有名管道两种，可以使用无名管道实现父子进程之间的通信。

首先使用 pipe()系统调用进行无名管道的创建。根据题目要求，使用 fork()系统调用创建两个子进程，用于向管道写数据。由于管道一般有两个端口（写口和读口），在某个进程中进行某项操作的时候，需要关闭其中一个口，并在另一个端口进行操作。因此，在子进程中，将读口关闭，在写口使用 write()系统调用，向管道写入预先定义好的字符串；在父进程中，将写口关闭，监听读口，使用 read()系统调用读管道中的数据。需要注意的是，由于管道被两个子进程写入，因此在读的时候需要读两次规定长度的字符串。

流程图如下：

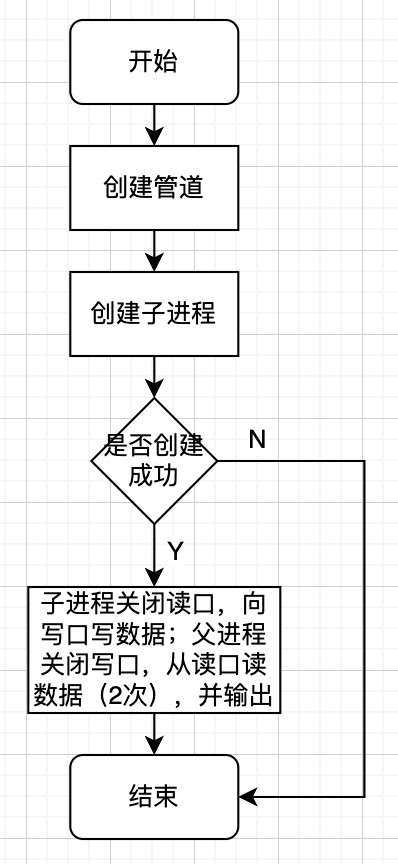


图 1 管道通信流程图

1. 实验结果与分析

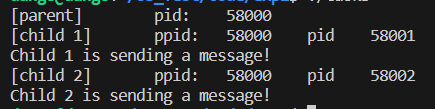


图 2 实验结果

如图2所示，可以看到程序成功创建了一个父进程和两个子进程。父进程的pid是58000，子进程的 pid 是 58001 和 58002。且可以观察到，两个子进程的 parent pid 均为 58000，说明这两个子进程确实是由同一个父进程创建出来的。

同时，在子进程创建后，子进程向管道中写入消息，父进程也成功的收到了。

# 实验三

1. 实验目的
2. 编写 Linux 环境下的多线程程序，了解多线程的程序设计方法，掌握最常用的三个函数 pthread\_create，pthread\_join 和 pthread\_exit 的用法；
3. 掌握向线程传递参数的方法。
4. 实验内容

使用两个线程实现数组排序和数组求和。

主程序中用数组 data[1000]保存 1000 个整数型数据（赋值为 1 到 10，循环 100 次），创建两个线程，一个线程将这个数组中的数据从大到小排列输出；另一个线程求出数组中所有数据的和。

1. 实验设计

首先了解三个函数的功能。pthread\_create 用于创建一个线程，pthread\_join 用于等待一个线程结束，pthread\_exit 用于退出。其次，还需要了解如何向线程传递参数：在 pthread\_create 中，又一个参数用于指定一个线程需要运行的函数，还有一个参数就是用于给需要执行的这个函数传递参数。

实验设计的需要注意的是，由于数组过长，可能导致一个线程在一个时间片内运行不完，即可能出现排序还未结束，求和就已经开始了。但是由于排序会导致数组元素位置的错乱，则有可能会导致求和时对同一个元素多次求和，导致最终结果的不正确。

因此，使用的方法为，在排序时使用选择排序，每次将排好的元素放到数组开头，并及时记录排好序的数组的最大下标。当切换到求和线程时，根据排好序的数组的最大下标位置进行求和即可。

流程图如下：

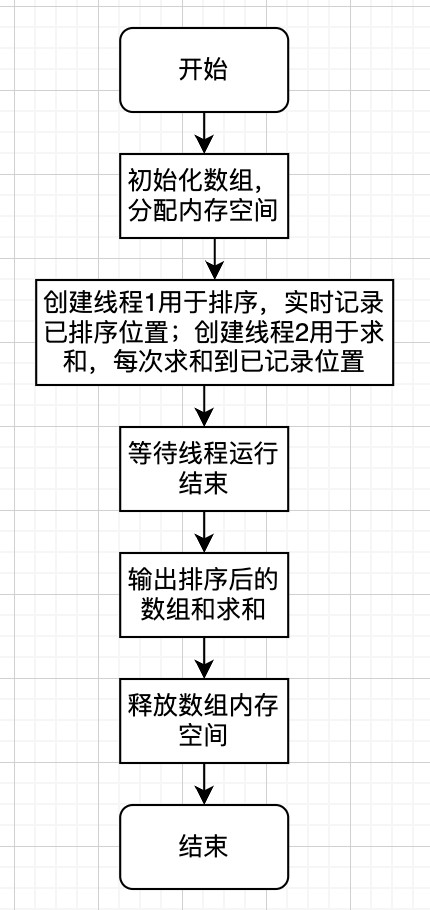


图 3 实验三 任务三

1. 实验结果与分析

如下图所示，可以看到最终排序和求和结果正确。

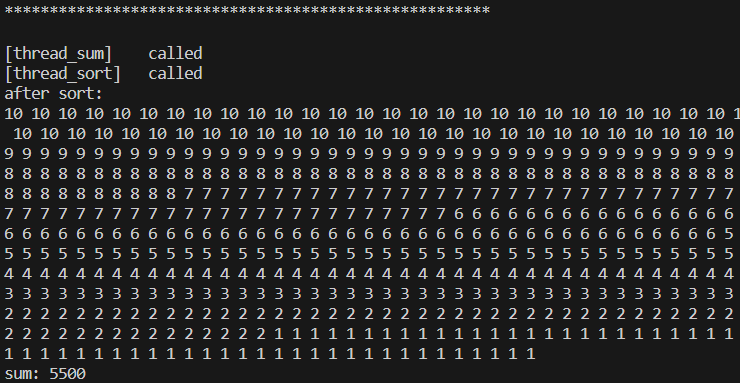


图 4 实验结果

# 实验四

1. 实验目的
2. 学习 UNIX 类（System V）操作系统信号量机制
3. 编写 Linux 环境下利用信号量实现进程控制，掌握相关系统调用的使用方法。
4. 实验内容
5. 生产者与消费者问题

有数据文件 1.dat 和 2.dat 分别存放了 10 个整数。创建 4 个线程，其中两个线程 read1 和 read2 负责分别从文件 1.dat 和 2.dat 中读取一个整数到公共的缓冲区，另两个线程 operate1 和 operate2 分别从缓冲区读取数据作加运算和乘运算。使用信号量控制这些线程的执行，保证缓冲区中的数据只能被计算一次（加或者乘），计算完成之后才能继续进行数据的读取工作。

提示：这是一个缓冲区长度为 2 的生产者消费者问题。read1 和 read2 是生产者，operate1 和 operate2 是消费者。需要互斥使用的缓冲区的长度为 2。在进行读取的时候不见得一定是 read1 读一个数然后 read2 读一个数，允许 read1 读得快（或者 read2 读得快），所以缓冲区中当前的数据都来自于 1.dat 或者 2.dat 是被允许的。计算也可能出现连续加和连续乘的情况。

例如，事先编辑好数据文件 1.dat 和 2.dat 的内容分别为 1 2 3 4 5 6 7 8 9 10 和 -1 -2 -3 -4

-5 -6 -7 -8 -9 -10，那么运行你编写的程序，可能得到如下类似的结果

1+-1=0

-2\*2=-4

3\*4=12

-3\*-4=12

5+-5=0

6\*-6=-36

…

1. 选做题

如果严格限制加和乘的两个操作数必须分别来自 1.dat 和 2.dat，且加法和乘法要严格交叉工作，应该如何修改上面的程序？运行结果为

1+-1=0

2\*-2=-4

-3+3=0

4\*-4=-16

5+-5=0

-6\*6=-36

…

1. 实验设计
2. 生产者与消费者问题

依题意可知，两个生产者不能同时向缓冲区写，两个消费者不能同时从缓冲区读。且生产者不能写满的缓冲区，消费者不能读空的缓冲区。每一个生产者每次生产1个数，每一个消费者每次消费2个数。

因此，设计了3个信号量。一个信号量用于记录缓冲区的容量，初值为2。一个信号量用于读的互斥，初值为1，最后一个信号量用于指示两个操作数是否读完，可以执行计算的操作，初值为0，这个信号量也同时实现了写的互斥。

流程图如下：

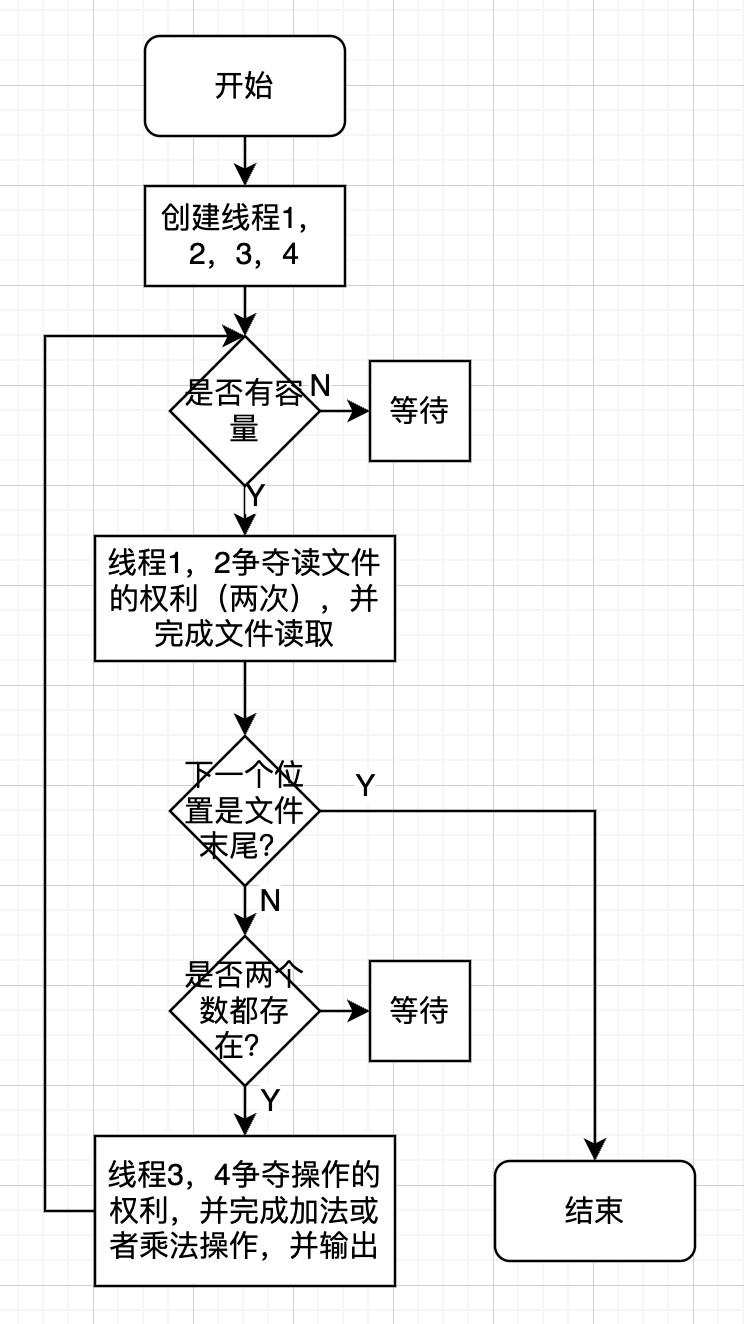


图 5 实验四 任务一

1. 选做题

依题意，需要严格限制加法和乘法的顺序，且两个操作数必须分别来自两个文件。因此定义4个信号量，使得他们可以按顺序执行。即，对于偶数次操作，规定流程为读1à读 2à加法；对于奇数次操作，规定流程为读1à读2à乘法。

使用1个信号量唤醒读1，使用1个信号量用于在读 1 完成后唤醒读 2。在读 2 运行时进行判断，如果是偶数次操作，就使用1个信号量唤醒加法操作；如果是奇数次操作，就使用1个信号量唤醒乘法操作。最后，分别在加法、乘法操作的末尾，使用唤醒读1的信号量，唤醒读1，开始新一轮的循环。

流程图如下：

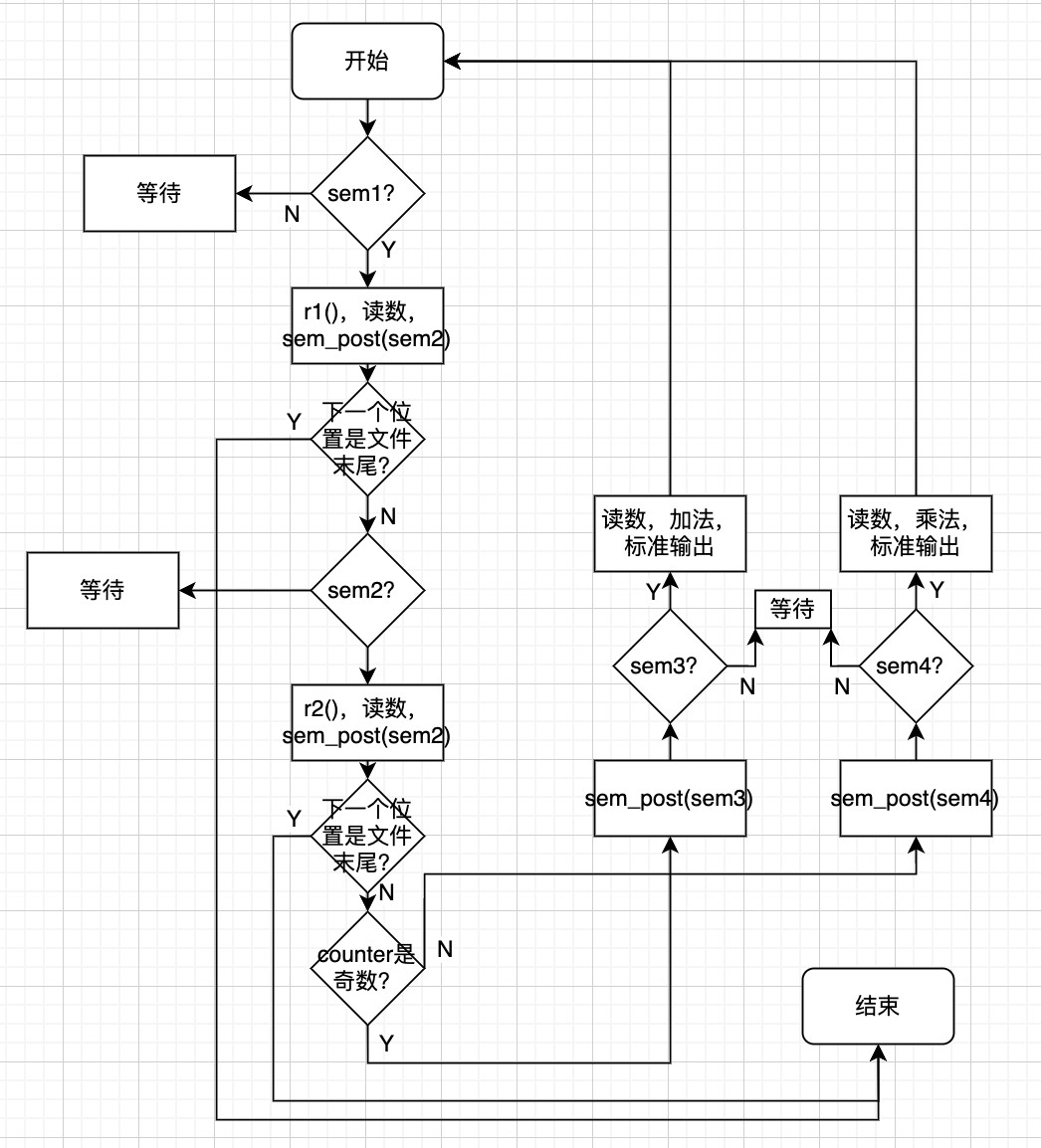


图 6 实验四 任务二

1. 实验结果与分析
2. 生产者与消费者问题

从结果可以看出每个线程之间都在争夺资源

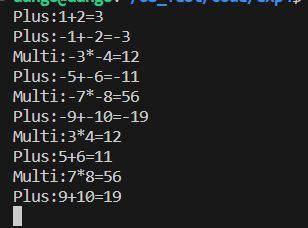


图 7 任务一

1. 选做题

从结果看出线程之间严格按照先读1，后读2，加法与乘法交叉执行

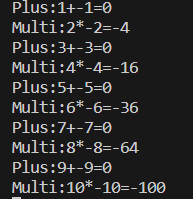


图 8 任务二

# 实验五

1. 实验目的

了解和熟悉 Linux 支持的消息通信机制及其使用方法

1. 实验内容

消息的创建、发送和接收

主函数中使用 fork()系统调用创建两个子进程 sender 和 receiver。sender 负责接收用户输入的一个字符串，并构造成一条消息，传送给 receiver。receiver 在接收到消息后在屏幕上显示出来。此过程一直继续直到用户输入 exit 为止。在程序设计过程中使用 msgget()、 msgsnd()、msgrcv()、msgctl()。

1. 实验设计

msgget()用于创建/获取一个消息队列，msgsnd()用于向指定的消息队列发送消息， msgrcv()用于从某个队列接收消息，msgctl()用于消息控制操作。

依据题意，首先使用 fork()系统调用创建两个子进程，并全局定义消息的结构体类型。在两个子进程中，一个用 msgget()创建一个消息队列，另一个用 msgget()获取到这个创建的队列。应该使用全局定义的 msqid 作为队列标识符。接着，在 sender 子进程中，使用 fgets() 从标准输入获取用户输入，新创建结构体类型变量以保存数据，使用 msgsnd()向消息队列发送数据，采用 IPC\_NOWAIT 策略。在 receiver 子进程中，使用 msgrcv()从消息队列接受消息，并使用 printf()进行标准输出。

需要注意的是，在 sender 子进程中，需要对用户的标准输入进行判断[3]。如果用户输入了”exit”，就使用 msgctl()、IPC\_RMID 参数删除指定消息队列，并使用 kill()系统调用杀死进程，程序结束。

流程图如下：

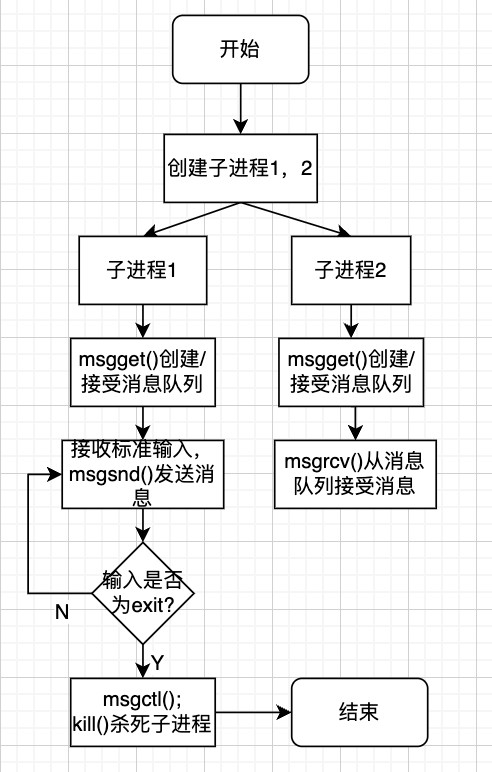


图 9 实验五流程图

1. 实验结果与分析

如图 10 所示，当输入字符串时，sender 接收并完成了消息发送，reveiver 从消息队列接受到了消息，并完成了标准输出。同时，当输入的字符串为 exit 时，进程被杀死，程序结束。

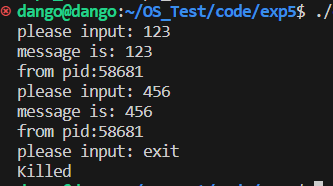


图 10 实验五结果

# 收获与体会

在完成操作系统实验的过程中，我深刻体会到了操作系统作为计算机系统的核心，对于并发编程和进程间通信等方面的重要性。通过使用Linux完成线程、信号量、管道、消息通信机制等实验内容，我获得了许多宝贵的经验和收获。

在线程方面，我学会了如何创建、同步和销毁线程。了解了线程的生命周期、线程的同步机制以及如何避免线程之间的竞态条件。这使我更好地理解了多线程编程的复杂性和挑战，同时也明白了如何通过合理的设计和同步手段来确保程序的正确性和稳定性。

通过实验使用信号量，我深入了解了进程间的互斥与同步。信号量作为一种重要的同步机制，对于解决进程之间的竞态条件、临界区问题等提供了有效的手段。这让我更加清晰地认识到在多任务环境中，如何通过信号量来协调进程的执行，确保资源的正确访问。

在管道和消息通信机制的实验中，我学到了不同进程间通信的方式。管道作为一种简单而有效的通信手段，可以在父子进程或者任意两个进程之间传递数据。而消息通信机制则更为灵活，通过消息队列等方式，实现了进程之间的异步通信，使得不同进程能够更加独立地执行任务，提高了系统的并发性和可扩展性。

总的来说，通过这一系列的实验，我不仅学到了操作系统底层机制的原理和实现方式，更加深入地理解了并发编程、进程间通信等概念。这些经验对于我未来在系统编程、并发控制、性能优化等方面的工作将会起到重要的指导作用。

在今后的学习中，我会继续尝试使用所学的知识解决新的问题，加深对知识的理解和掌握。同时，尝试拓展自己的知识面，学习新的知识和技能。例如，我可以通过学习和实践相结合的方式，深入了解计算机系统的工作原理，或者学习一些其他相关的专业知识。

总之，学习是一个持续不断的过程，要想在计算机领域取得成功，就需要不断学习和提高自己的能力，持续不断的探索未知领域，终身学习。

# 参考资料

[1]实验指导书