操作系统实验报告

实验一：进程管理

姓名：胡敏臻

学号：2019211424

班级：2019211307

日期：2021/11/20

**摘要**：本次实验完成了读者-写者实验中的修改输入，在修改输入的过程中发现了原文件在读者优先和写者优先中的bug。因此在所给文件基础上进行修改，修复了目前发现的bug。接着完成了生产者-消费者实验，完成的实验任务为课堂讲解的一道例题，家长给小孩分水果，小孩从盘子中拿水果的过程。

## 读者-写者实验

一、实验目的

本实验旨在让学生动手设计一个进程同步互斥的实验，更深刻的理解进程协作机制。

openEuler 提供了互斥量(pthread\_mutex\_t)、信号量(sem\_t)等同步对象和相应的系统调用，用于线程的互斥与同步。学生可通过对读者写者问题的调试或者生产者消费者问题的调试（选做其一即可），了解 openEuler 中的同步、互斥机制。

二、实验描述

利用 openEuler 信号量机制，实现读者写者问题的模拟。

在 openEuler 环境下，创建一个包含 n 个线程的控制进程。用这 n 个线程来表示 n 个读者或写者。每个线程按相应测试数据文件的要求，进行读写操作。请用信号量机制分别实现读者优先和写者优先的读者-写者问题。

读者-写者问题的读写操作限制：

1）写-写互斥；

2）读-写互斥；

3）读-读允许；

读者优先的附加限制：如果一个读者申请进行读操作时已有另一读者正在进行读操作，则该读者可直接开始读操作。

写者优先的附加限制：如果一个读者申请进行读操作时已有另一写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。

运行结果显示要求：要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息，以确信所有处理都遵守相应的读写操作限制。

三、实验要求及内容

3.1 实验具体内容

复现代码，并修改输入数据，分析结果。

3.2 实验原理

**读者优先**

在 readFirst.c 中，首先读取目标文件 test，为每一行请求创建一个线程，其中读请求创建读者线程，写请求创建写者线程，调用 pthread\_create()函数。引入计数器 reader\_count 对读进程计数，writer\_count 对写进程计数。 mutex 是互斥信号量。

读者优先需要引入一个状态变量 state，表明当前系统的状态。需要用到的几个状态量如下：

• mutex：保证 readercount, writercount, state 变量访问的互斥性

• Sig\_read：允许读的信号量

• Sig\_wrt：允许写的信号量

读者优先的设计思想是读进程只要看到有其它读进程正在读，就可以继续进行读；写进程必须等待所有读进程都不读时才能写，即使写进程可能比一些读进程更早提出申请。该算法只要还有一个读者在活动，就允许后续的读者进来，该策略的结果是，如果有一个稳定的读者流存在，那么这些读者将在到达后被允许进入。而写者就始终被挂起，直到没有读者为止。 **写者优先**

在 writeFirst.c 中，首先读取目标文件 test，为每一行请求创建一个线程，其中读请求创建读者线程，写请求创建写者线程，调用 pthread\_create()函数。引入计数器 reader\_count 对读进程计数，writer\_count 对写进程计数。 mutex 是互斥信号量。

写者优先需要引入一个状态变量 state，表明当前系统的状态。需要用到的几个状态量如下：

• mutex：保证 readercount, writercount, state 变量访问的互斥性

• Sig\_read：允许读的信号量

• Sig\_wrt：允许写的信号量

写者优先的设计思想是在一个写者到达时如果有正在工作的读者，那么该写者只要等待正在工作的读者完成，而不必等候其后面到来的读者就可以进行写操作。该算法当一个写者在等待时，后到达的读者是在写者之后被挂起，而不是立即允许进入。

四、实验设备环境

openEuler 服务器和虚拟机。

五、实验步骤

1、修改输入test,修改如下：

1 R 2 2

2 W 3 4

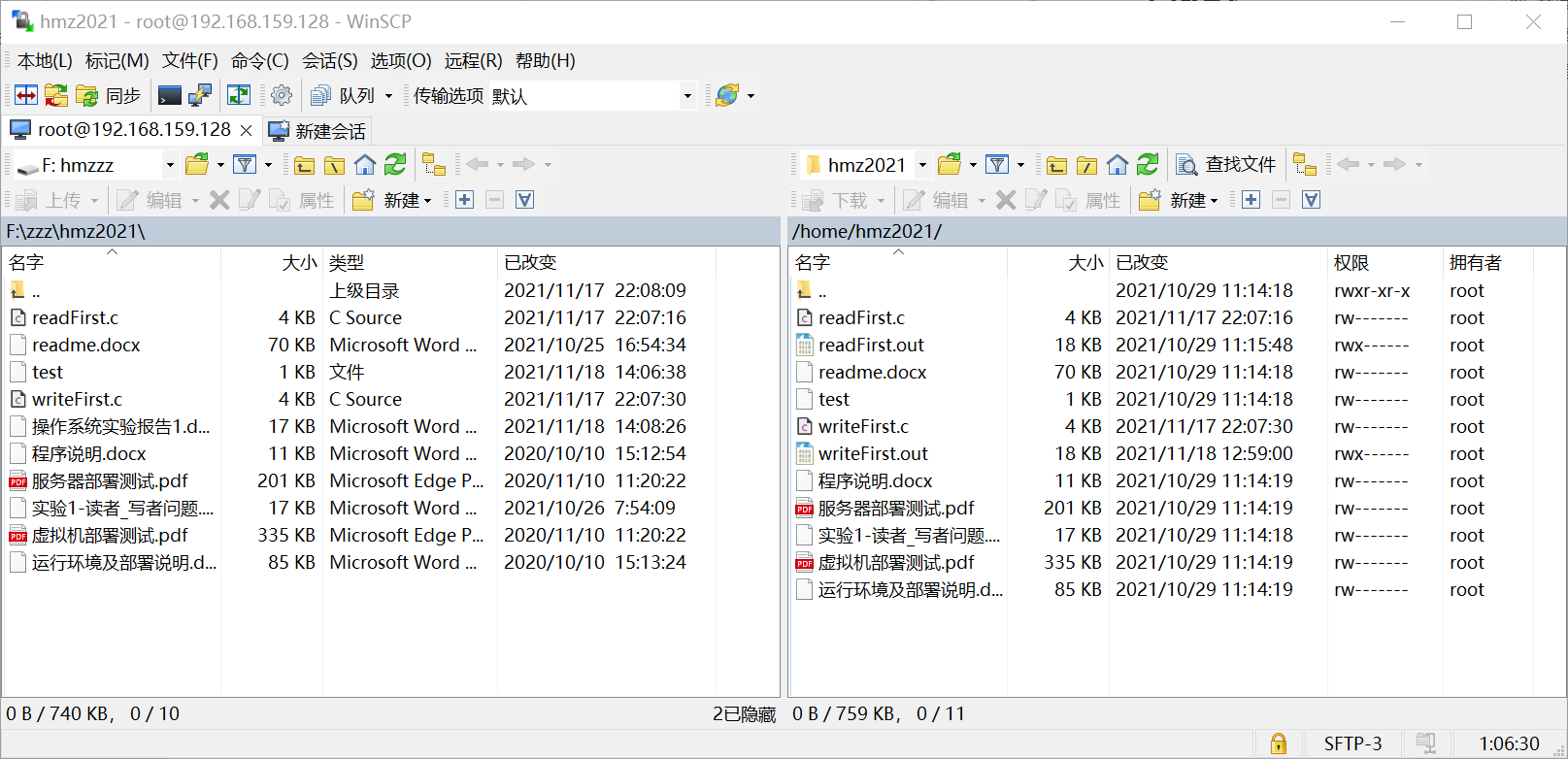
3 W 4 2

4 R 5 3

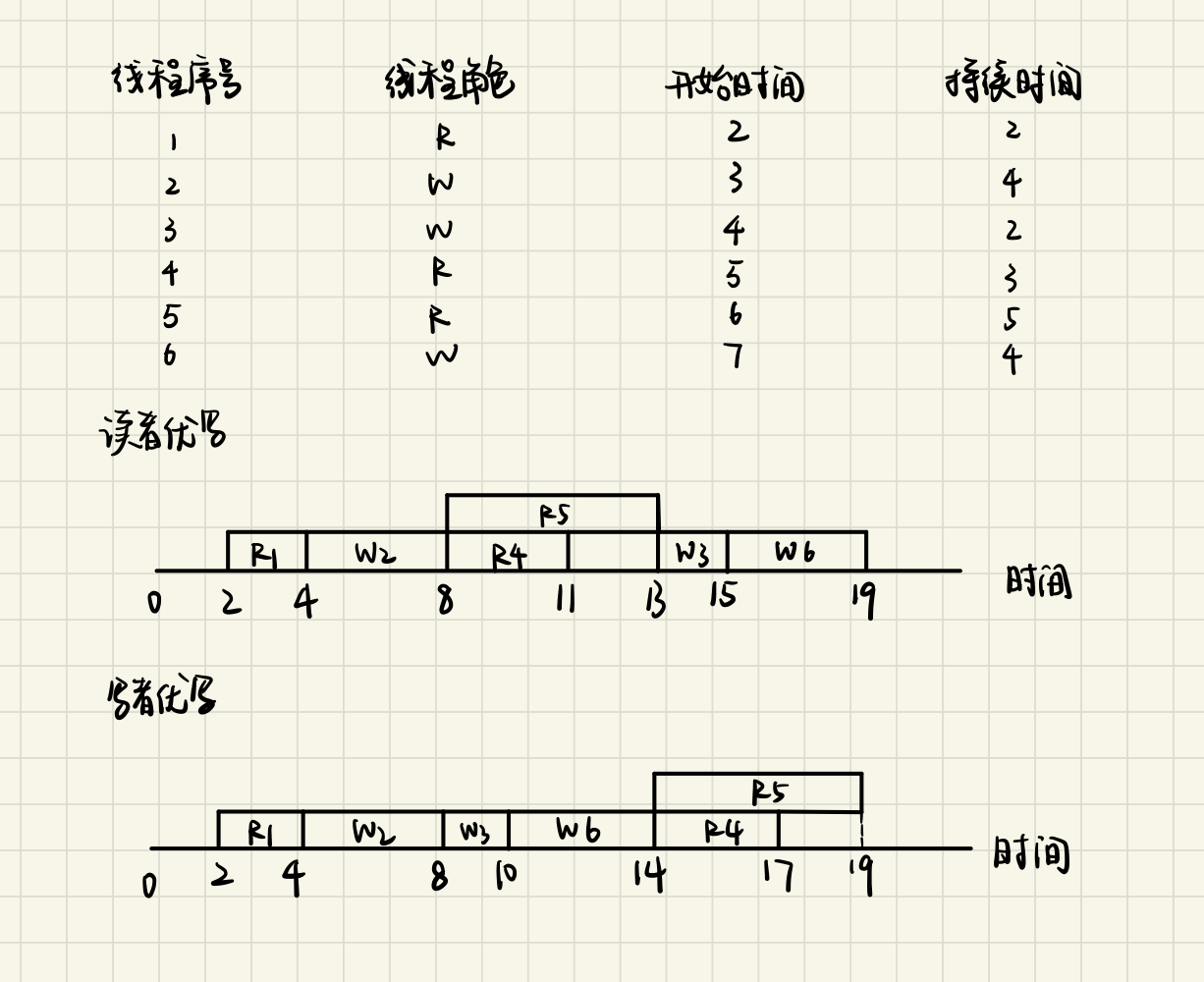
5 R 6 5

6 W 7 4

2、利用winSCP连接到服务器，将文件和test文件上传



3、进行理论分析



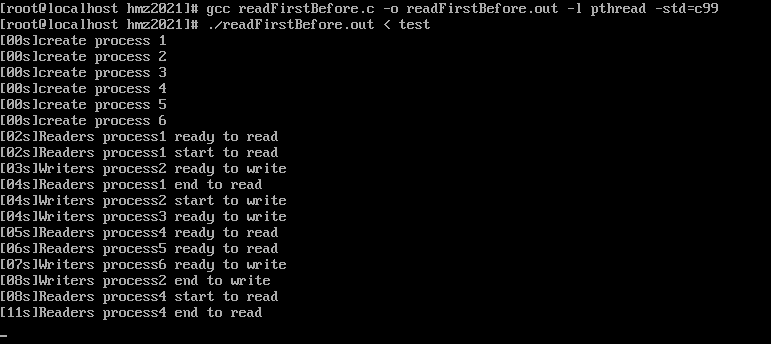
4、首先编译读者优先，进行运行

编译：gcc readFirst.c -o readFirst.out -l pthread -std=c99

运行：./readFirst.out < test

5、调试程序bug

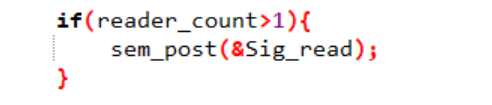
运行之后真是给我了一个大大的“惊喜”，我竟然发现程序bug了呀，在我这个输入时，程序会在11s时被卡住，如下：



后来与理论分析后得知，发现在第8秒的时候r5没有start to read，根据分析得知，此时因为写者进程执行完之后只会释放一个sig\_read，因此r5就没有机会执行了。这也将导致read\_count一直大于0，sig\_write信号也没法释放，因此整个程序就会卡住了。因此，我们需要解决的问题在于：怎么在写者进程进来的读者进程可以给到正确的sig\_read？

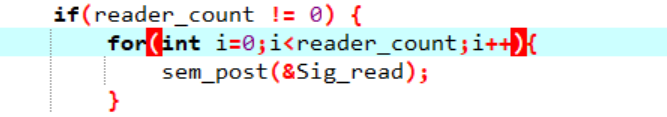
6、关于读者进程怎么修改，目前想到了两种修改的方式，

一是在等到sig\_read之后，判断现在的read\_count是否大于1，若大于1的话则增加sig\_read信号，其中需要注意的是，如果读者进程执行完了，需要sig\_read信号清0，防止影响到后面的进程。



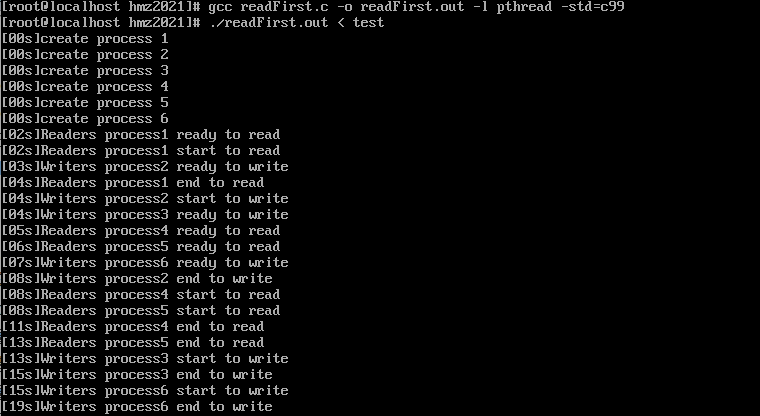


二是在写者进程执行完之后，发现有几个read\_count就释放几个read\_count。



（后来发现这一点在写者优先的程序中其实有哦）

修改后的结果为：



程序正常运行完毕。用这两种方法都可以解决问题。

7、后来咱们开始写者优先，好家伙，不运行不知道，一运行吓一跳，原来写者优先也有bug啊。刚开始还是老一套：

gcc writeFirst.c -o writeFirst.out -l pthread -std=c99

./writeFirst.out < test

还是运行修改后的输入发现，读者-写者不互斥了。

这次的输入为：

1 W 2 3

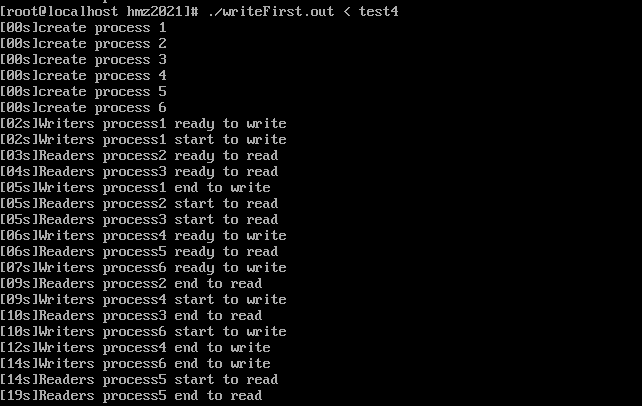
2 R 3 4

3 R 4 5

4 W 6 3

5 R 6 5

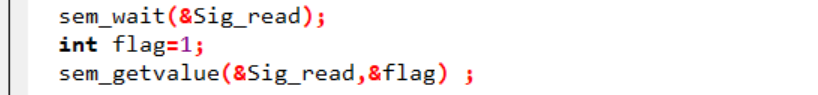
6 W 7 4

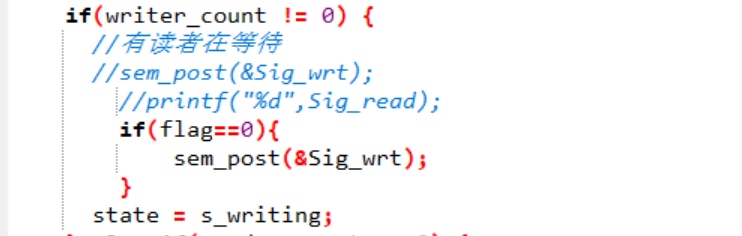


分析：09秒的时候W4已经start to write了，而R3是从05秒开始到10秒结束，这里存在了读写一起运行的现象了。并且W6在第10秒开始，而W4在12秒才结束，这时候也存在着写写不互斥了。

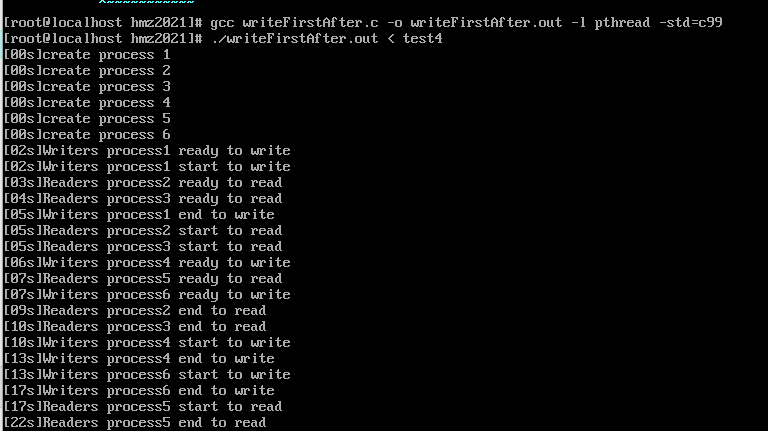
导致这个现象的原因是因为在某一个读者进程结束后，就会释放sig\_write的信号，而此时可能还存在着其他的读者进程还没有结束，此时就会发生读和写同时存在。而之后结束的读者进程又会释放sig\_write的信号，导致后面的写者可以运行了，因此这也导致了写与写的同时存在。

8、关于写者优先怎么改。基本的想法是需要在当前运行的所有读者进程运行完之后才能进行写者进程。因此当读到有写者进程时，只切换状态，而不释放信号。那么如何标识当前运行完了。在这里想了很多办法也踩了很多坑。一开始判断当此时的读者进程数量为0时，释放sig\_wrt信号。但是这里也有个问题。如果此时后面又有读者进程，此虽然是后来的，但是它还是让读者进程数量增加了，而前面真正的读者进程已经运行结束了，此时却没有捕捉到信号，因此又会产生错误。于是想了一个办法，在wait到sig\_read之后，判断信号量的值是否为0，将值赋值给flag，为0就说明此时没有需要运行的读者进程，就说明此进程已经是当前运行的最后一个进程，在最后判断这个进程的flag，如果为0就释放sig\_wrt。按照如上要求，对程序改动如下：





可喜可贺，此时程序终于可以正常运行了。运行结果如下：



看，此时正常运行了，读-写互斥与读-读互斥都正确了。感动子;)

## 生产者-消费者实验

一、实验目的

本实验旨在让学生动手设计一个进程同步互斥的实验，更深刻的理解进程协作机制。

openEuler 提供了互斥量(pthread\_mutex\_t)、信号量(sem\_t)等同步对象和相应的系统调用，用于线程的互斥与同步。学生可通过对读者写者问题的调试或者生产者消费者问题的调试（选做其一即可），了解 openEuler 中的同步、互斥机制。

二、实验描述

利用 openEuler 信号量机制，实现生产者消费者问题的模拟。

在 openEuler 环境下，创建一个包含 n 个线程的控制进程。用这 n 个线程来表示 n 个生产者或消费者。每个线程按相应测试数据文件的要求，进行读写操作。请用信号量机制实现生产者-消费者问题。

三、实验要求及内容

在该实验中，构造生产者与消费者进程，实现任意一功能即可。

在本实验中，sig\_plate标识盘子的锁，sig\_apple标识苹果的锁。只有在盘子为空时，Dad才能放水果，只有当盘中有苹果时，Daughter才能吃到苹果。

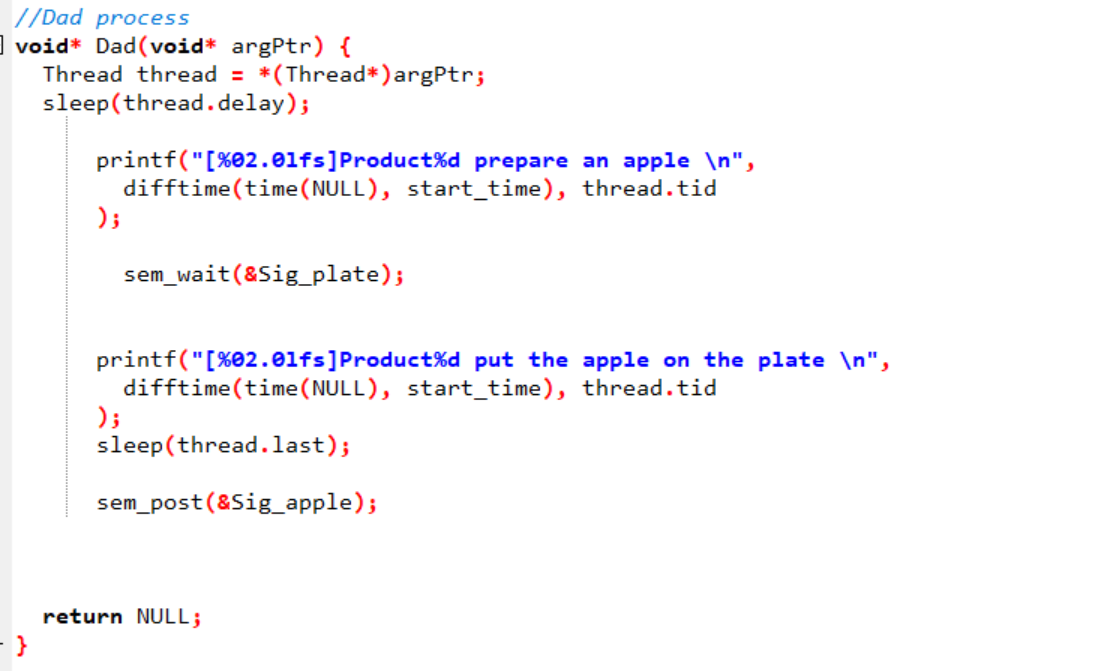
四、实验设备环境

openEuler 服务器和虚拟机。

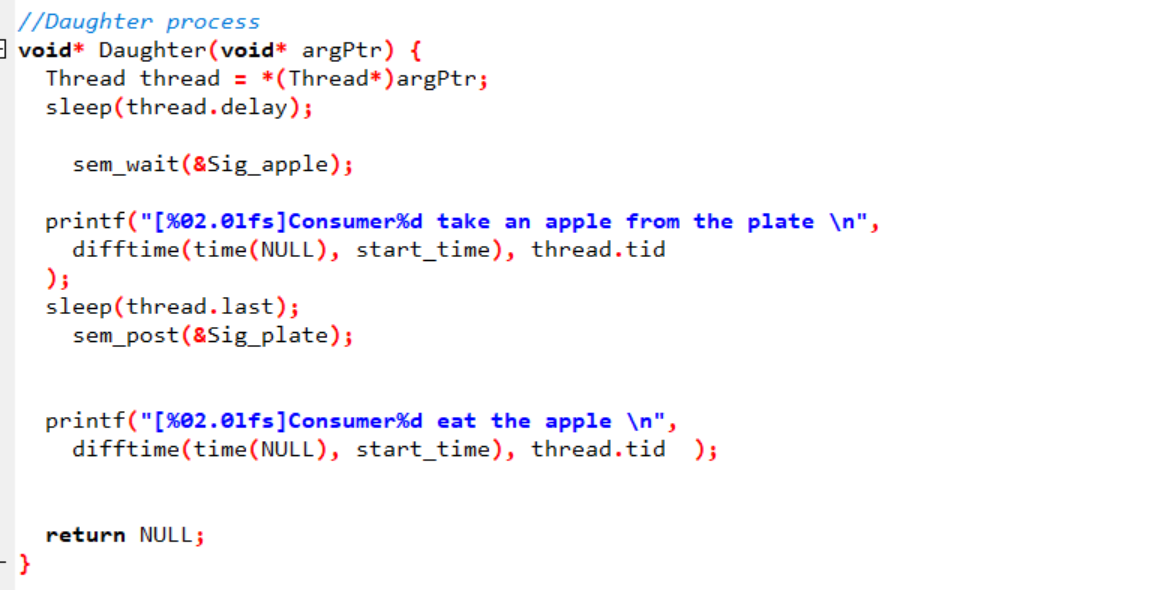
五、实验步骤

1、按照要求编写程序

Dad是生产者进程，生产苹果：



Daughter是消费者进程，吃苹果：



2、编写输入，仿照读者-写者进程，模仿输入：

1 P 2 2

2 C 3 4

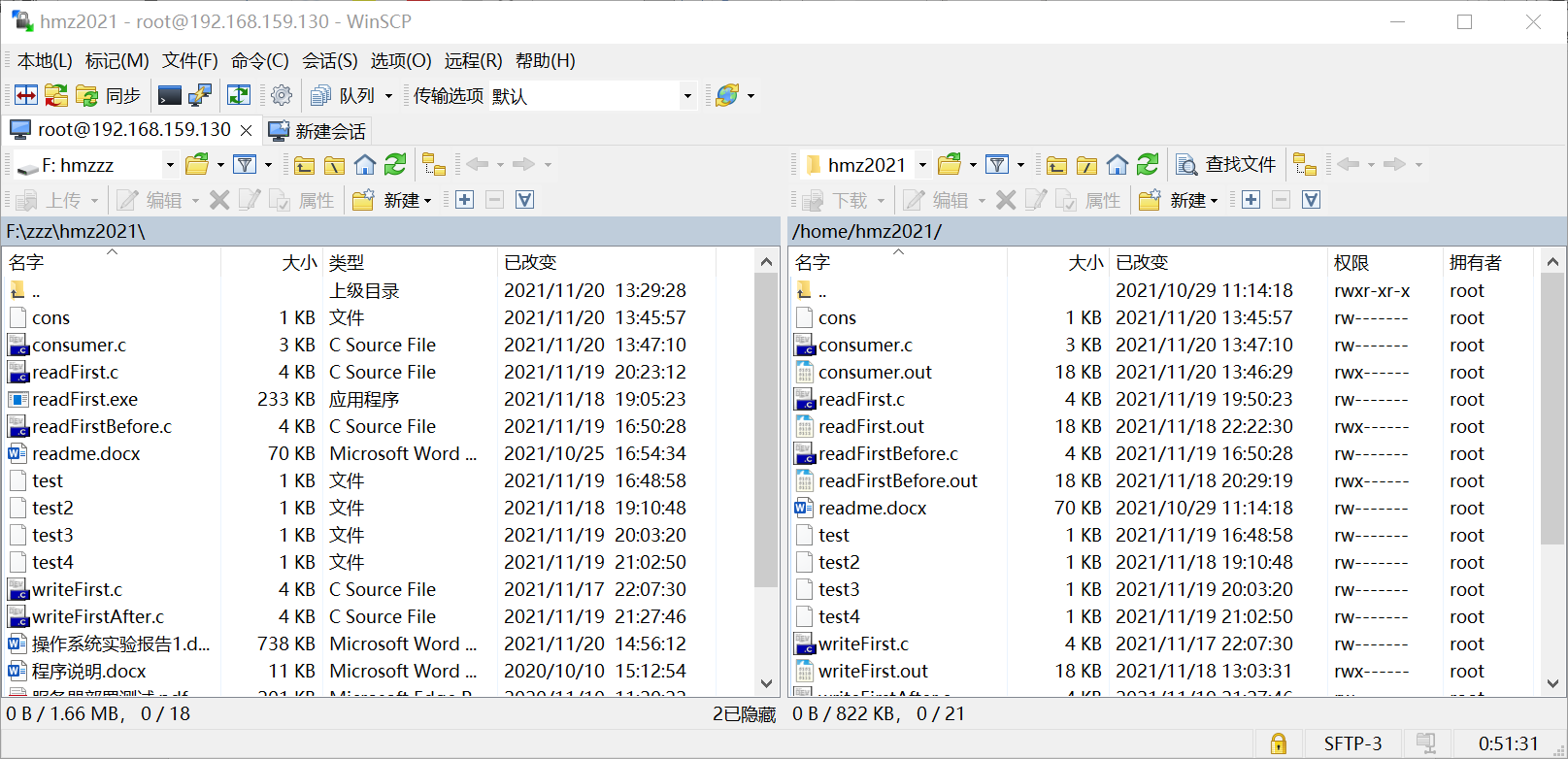
3 C 4 2

4 P 5 3

5 P 6 5

6 C 7 4

3、将consumer.c与cons上传到虚拟机中

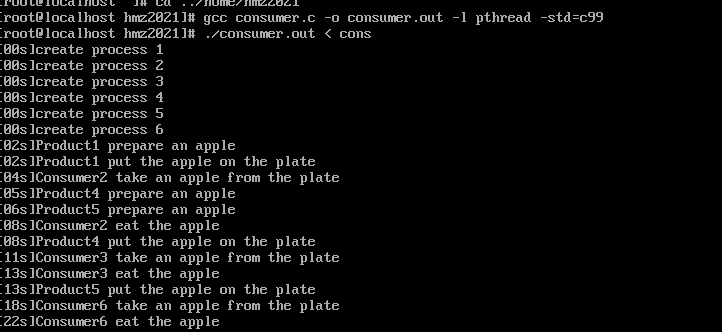


4、编译程序并运行

编译：gcc consumer.c -o consumer.out -l pthread -std=c99

运行：./consumer.out < test

5、结果展示



发现程序正常运行，此时只允许有一个生产者和一个消费者，发现程序会被sig\_plate和sig\_apple锁住，因此只有一个Dad允许放苹果，也只有一个Daughter可以吃到苹果，查看运行结果，符合预期。

## 实验总结

实验时间：8个小时

实验心得：本次实验所花费时间较长，主要时间花费在了对读者优先和写者优先两个程序的debug上。一开始以为这个实验修改输入完成应该是比较顺利的事情。但是在修改输入的过程中发现读者优先和写者优先都存在着问题。修改程序这个过程其实还挺考验人的，因为你要找出是在哪个过程出现了问题，并且需要想办法怎么样修改这个bug。在这个过程其实可以不断对程序有更深的理解和思考，在分析之后得到是什么地方出了错误，并且可以想到要怎么修改。修改的过程其实也不是一帆风顺的，并不一定你修改以后程序就能正常运行的。比如一开始修改读者优先的代码，打算判断如果wait之后read\_count计数器大于1就继续释放信号，但这种情况必然会导致信号的冗余，如果程序继续运行，就会被冗余的信号干扰了。所以后来想到如果运行完了信号量要清零。在写者优先进程我们也看到了解决的更好办法，就是在写者进程执行完后，释放正确的信号量数量。同样在写者优先，修改程序的时候也遇到了问题。一开始判断read\_count为0时释放，可是后来发现，可能read\_count不为0时也要释放sig\_wrt。关于怎么判断此时读者进程结束了，最后也是尝试了很久，才决定在wait之后判断信号量是否为0来判断此时读者进程有没有结束，最后成功修改了bug。因为修改一个地方其实要考虑很多方面，需要做细致周全的考虑，如果疏忽了某一点，程序可能就运行不对了，在不断尝试修改bug的过程中，其实也对读者-写者进程会有更加深入的理解，考虑不同情况的时候也会思考的更多一点。最后终于解决了bug的时候也是非常有成就感的。

后来又开始编写了消费者与生产者进程实验，突然想到那个苹果的例题，比较简单而且也比较有意思，刚学习这块内容的时候，这个例题帮助我理解了进程运行的情况，于是编写了这个程序。因为前面有了读者-写者实验的铺垫，因此完成这个实验的过程反倒较为顺利。并且这个实验相较前者更容易实现，没有那么多的限制，实现起来也比较容易，因此最后也顺利完成了。

总的下来，本次实验还是收获了很多，不过本次实验花费的时间也挺多的。从安装openEuler，到学习调试环境，到运行程序修改代码，到自己写代码，一步步慢慢深入，时间跨度也比较长，也说明对进程管理有了更深入的理解了。老师对我们也没有什么限制，我们也可以自己发挥，看到一个问题的时候努力搞清楚并解决，整个实验过程还是非常有趣的。

辛苦老师和助教了,祝好！