西 安 邮 电 大 学

（计算机学院）

课内实验报告

实验名称：进程/线程同步

专业名称：软件工程

班　　级：软件 1901

学生姓名：范佳伟

学号（8 位）：04193013

指导教师：陈莉君

实验日期：2021 年　10 月　14 日

一. 实验目的及实验环境

1.实验目的

通过观察、分析实验现象，深入理解理解信号量的原理及特点，并掌握在 POSIX 规范中的信号量的功能及使用方法。

2.实验环境

（1）硬件

- CPU：Pentium111 以上；
- 内存：128MB 以上；
- 显示器：VGA 或更高；
- 硬盘空间：至少 100MB 以上剩余空间。

（2）软件

- 虚拟机名称及版本：VMware9.0.2
- 操作系统名称及版本：Linux version 4.15.0-47-generic (buildd@lcy01-amd64-016)
- 编译器：Ubuntu 下 gcc 编译器。

二. 实验内容

1、实验前准备工作

阅读参考资料，了解信号量机制的实现原理，并对生产着消费者问题，读者写者问题，哲学家就餐问题进行深刻理解。

2、实验内容

以下内容至少选择一种完成（鼓励全部完成，会有不一样的体会）

1）从对生产者消费者问题，视频给出几种方案，可逐步实现，并对其进行比较，有的同学已经实现了这种代码，就不用做了。

2）读者写者问题，从读者优先到写者优先，给出解决方案，并进行分析

3）哲学家就餐问题，从基本方案到解决死锁的方案，并进行分析

3、提问并回答

提出至少两个问题，并给予回答，或同组内，在讨论区，两个同学为一组，一个提问，一个回答。

读者-写者问题：

信号量机制问题：

一个共享数据区，有若干个进程负责对其进行读入工作，若干个进程负责对其进行写入工作。

1、允许多个进程同时读数据

2、互斥读数据

3、若有进程写数据，不允许读者读数据

解决办法：

读者优先

基本描述:一旦有读者正在读数据，允许多个读者同时进入读数据，只有当全部读者退出，才允许写者进入写数据。

满足条件：

对于读者：

没有读者和写者，直接读数据；

有写者等待，其他读者正在读，越过写者，直接进入读数据；

有写者写数据，新读者等待

对于写者：

无读者，新写者直接写；

有读者，新写者等待；

有写者，新写者等待

问题 1：什么是进程同步，为什么要引入进程同步？

进程同步是指对多个相关进程在执行次序上进行协调，以使并发执行的主进程之间有效地共享资源和相互合作，从而使程序的执行具有可在现行。 多个线程同时访问共享资源时候,如果没有先来后到,可能造成结果的不可再现性,使得 结果背离预期的结果,例如多人抢少量票,银行取钱等.

问题 2：同步机制应该遵循的原则是什么？为什么要遵循这些原则？

1.空闲让进。并发过程中某个进程不在临界区时，不阻止其它进程进入临界区 2.忙则等待。并发进程中，若干个进程申请进入临界区时，只允许一个进程进入。当已 有进程进入临界区时，其它申请进入临界区的进程必须等待，以保证对临界资源的互斥访问。 3.有限等待。访问临界资源的进程应保证在有限的时间内进入自己的临界区，避免因长 时间申请临界资源得不到满足，而一直等待下去，陷入“等死”状态。 4.让权等待。当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙 等”状态

三．方案设计

读者-写者问题：

实现思路：

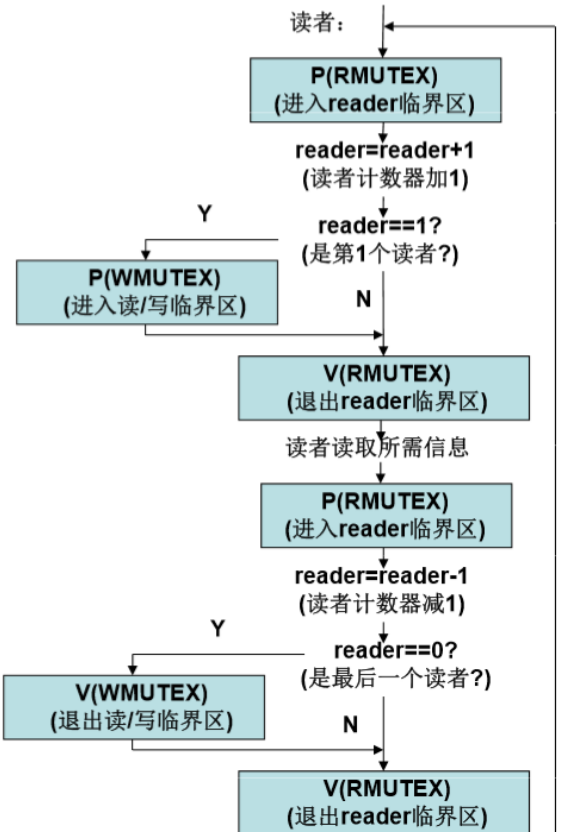
设置一个共享变量和两个信号集

共享变量 reader：记录当前正在读数据集的读进程数，初值为 0

读互斥信号量 rmutex：表示读进程互斥访问共享变量 reader，初值为 1，互斥信号

写互斥信号量 wmutex：表示写进程与其他进程（读写进程）互斥访问数据集，初值为 1. 互斥信号量

附流程图:



四．测试数据、运行结果以及调试过程截图

```
Thread-3 be ready to write data!
Thread-3 have write data: 2491
Thread-3 be ready to write data!
Thread-3 have write data: 1105
Thread-3 be ready to write data!
Thread-3 have write data: 5466
Thread-3 be ready to write data!
Thread-3 have write data: 1467
Thread-3 be ready to write data!
Thread-3 have write data: 4198
Thread-3 be ready to write data!
Thread-3 have write data: 3457
```

五．总结

1． 实验过程中遇到的问题及解决办法；

1.对于计数器的访问，一定是一个一个的访问，一个一个的数，互斥访问计数器。

2.互斥访问计数器，wait（rmutex）还可以阻塞后面读进程，如果当前的读进程因为有写进程陷入了阻塞，后面的读进程进不了计数器，就陷入了阻塞。如果没有写进程，还可以阻塞后来的写进程

3.同时通过绕过 wait（S）操作，就不存在互斥访问了；绕过进入区，随便访问临界资源

2．对设计及调试过程的心得体会。

心得：互斥访问计数器的时候必须要是一个同一个资源信号量，在进入时没有释放，就不能退出，在退出没有释放时，就不能进入。防止出现同时写入和退出，造成判断条件的混乱通过将 p 操作有不同的进程操作，实现进程之间的相互控制，比如上述，通过写者线程控制读者线程，实现了写者优先。

六．附录：源代码（电子版）

```
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

@Slf4j
public class ReaderWriterTest {

    static Semaphore rMutex = new Semaphore(1);
    static Semaphore wMutex = new Semaphore(1);
    static int readCount = 0;

    static class Reader extends Thread {
        Reader(String name) {
            super.setName(name);
        }

        @Override
        public void run() {
            do {
                try {
                    rMutex.acquire();
```

```
            if(readCount == 0){
                wMutex.acquire();
            }
            readCount ++;
            //log.info("读者【{}】在读操作执行结束，当前读者数：【{}】", readCount);
            rMutex.release();
            log.info("读者【{}】在执行读操作,当前读者数：【{}】", getName(), readCount);

            Thread.sleep(5000);
            rMutex.acquire();
            readCount --;
            if(readCount == 0){
                wMutex.release();
            }
            rMutex.release();
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            log.error("哲学家执行时产生异常！ ");
        }
    } while (true);
  }
}

static class Writer extends Thread {
  Writer(String name) {
    super.setName(name);
  }

  @Override
  public void run() {
    do {
      try {
        wMutex.acquire();
        log.info("写者【{}】执行了写操作", getName());
        Thread.sleep(1000);
        wMutex.release();
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        log.error("写进程执行时产生异常！ ");
      }
    } while (true);
  }
}

public static void main(String[] args) {
  Reader r1 = new Reader("r1");
  Reader r2 = new Reader("r2");
  Reader r3 = new Reader("r3");

  Writer w1 = new Writer("w1");
  Writer w2 = new Writer("w2");

  r1.start();
  r2.start();
  r3.start();
  w1.start();
  w2.start();
```

```
    }
}
```