嵌入式软件开发技术与工具实验报告三

黎炜桁

一、实验目的

- 掌握基于pthread线程库的多线程编程技术。
- 掌握基本的线程间同步技术(sem，mutex)。
- 理解共享资源并掌握其操作方法。

二、实验内容

1. 读者-写者问题多线程实现
    - 一个数据集（如数据、文件等）被N个线程读写；
    - 一些线程只要求读数据集内容，称为读者 (Reader)，实验读者数不少于6个；
    - 另些线程要求修改数据集内容，称为写者 (Writer)，实验写者数不少于3个；
    - 多个读者可以同时读数据集内容，不需要互斥操作；
    - 一个写者不能和其他写者或读者同时访问数据集，换句话说，写者和其他写者或读者之间必须互斥操作！
    - 读者优先：如果有读者，写者需要等待！用于更新不频繁或更新影响不显著、读者较多的场合；
    - 写者优先：如果有写者，读者需要等待！用于更新频繁或更新影响显著的场合。

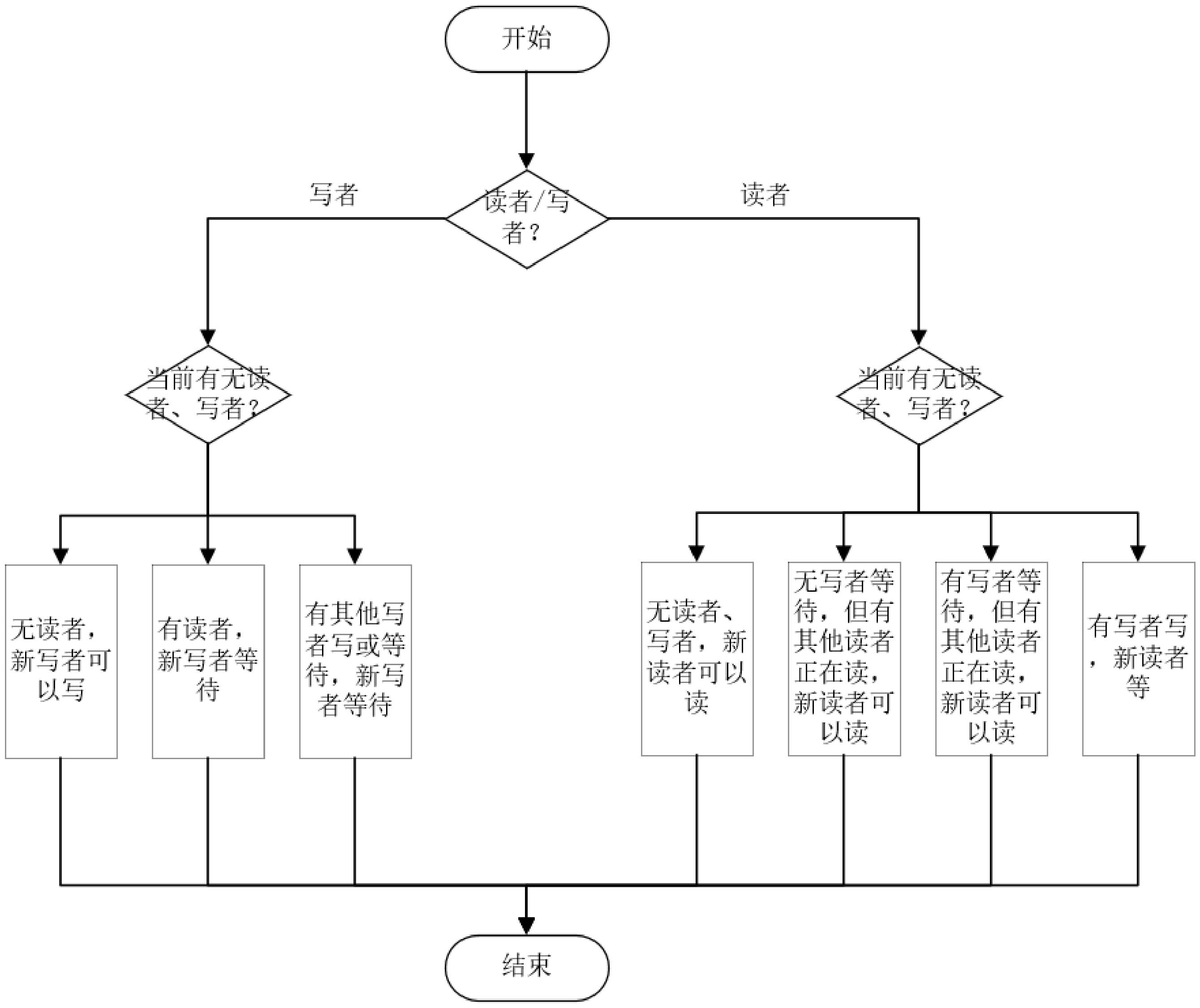
三、实验过程与结果

1. 读者优先

读者优先描述
如果读者来：

- 无读者、写着，新读者可以读；
- 无写者等待，但有其他读者正在读，新读者可以读；
- 有写者等待，但有其他读者正在读，新读者可以读；
- 有写者写，新读者等
  如果写者来：
- 无读者，新写者可以写；
- 有读者，新写者等待；
- 有其他写者写或等待，新写者等待

读者优先算法流程图如下

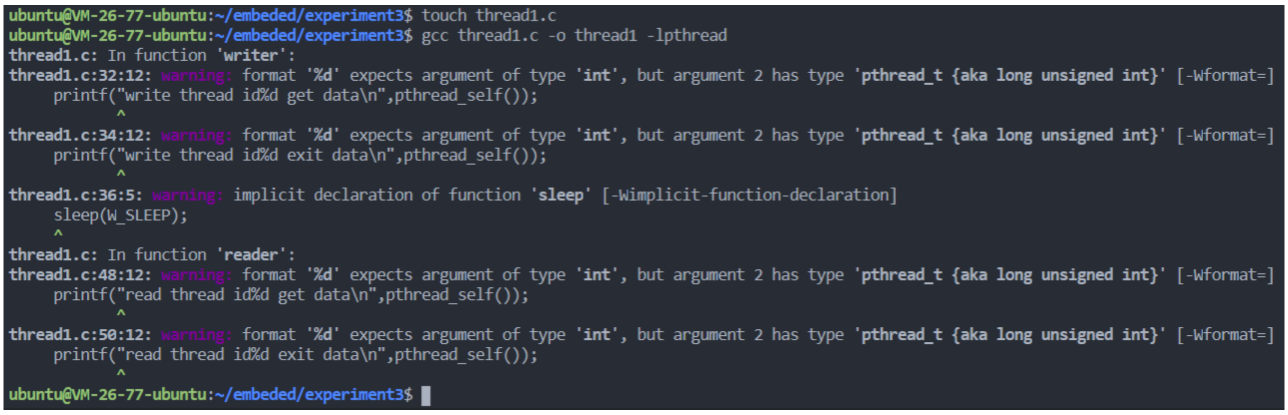


- thread1.c

```
#include "stdio.h"
#include <stdlib.h>
#include <pthread.h>
#include<semaphore.h>


#define N_WRITER 30 //写者数目
#define N_READER 5 //读者数目
#define W_SLEEP  1 //控制写频率
#define R_SLEEP  1 //控制读频率


pthread_t wid[N_WRITER],rid[N_READER];
pthread_mutex_t mutex_write;//同一时间只能一个人写文件,互斥
sem_t sem_read;//同一时间只能有一个人访问 readerCnt
int data = 0;
int readerCnt = 0;
void write()
{
    int rd = rand();
    printf("write %d\n",rd);
    data = rd;
}
void read()
{
    printf("read %d\n",data);
}
void * writer(void * in)
{

    pthread_mutex_lock(&mutex_write);
    printf("write thread id%d get data\n",pthread_self());
    write();
    printf("write thread id%d exit data\n",pthread_self());
    pthread_mutex_unlock(&mutex_write);
    sleep(W_SLEEP);
    pthread_exit((void *) 0);
}

void * reader (void * in)
{
    sem_wait(&sem_read);
    readerCnt++;
    if(readerCnt == 1){
        pthread_mutex_lock(&mutex_write);
    }
    sem_post(&sem_read);
    printf("read thread id%d get data\n",pthread_self());
    read();
    printf("read thread id%d exit data\n",pthread_self());
    sem_wait(&sem_read);
    readerCnt--;
    if(readerCnt == 0){
        pthread_mutex_unlock(&mutex_write);
    }
    sem_post(&sem_read);
    sleep(R_SLEEP);
    pthread_exit((void *) 0);
}

int main()
{
    printf("read first\n");
    pthread_mutex_init(&mutex_write,NULL);
    sem_init(&sem_read,0,1);
    int i = 0;
    for(i = 0; i < N_WRITER; i++)
    {
        pthread_create(&wid[i],NULL,writer,NULL);
    }
        for(i = 0; i < N_READER; i++)
    {
        pthread_create(&rid[i],NULL,reader,NULL);
    }
    sleep(1);
    return 0;
}
```

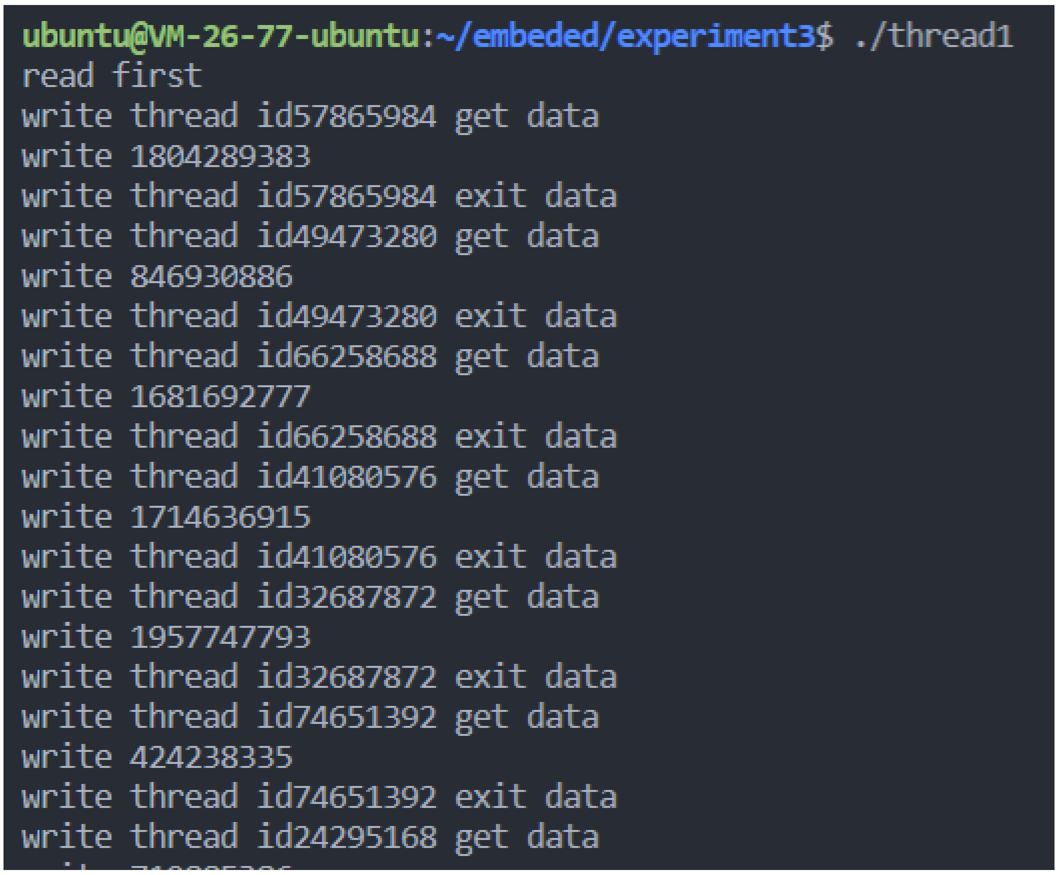
- 实验结果

编译文件

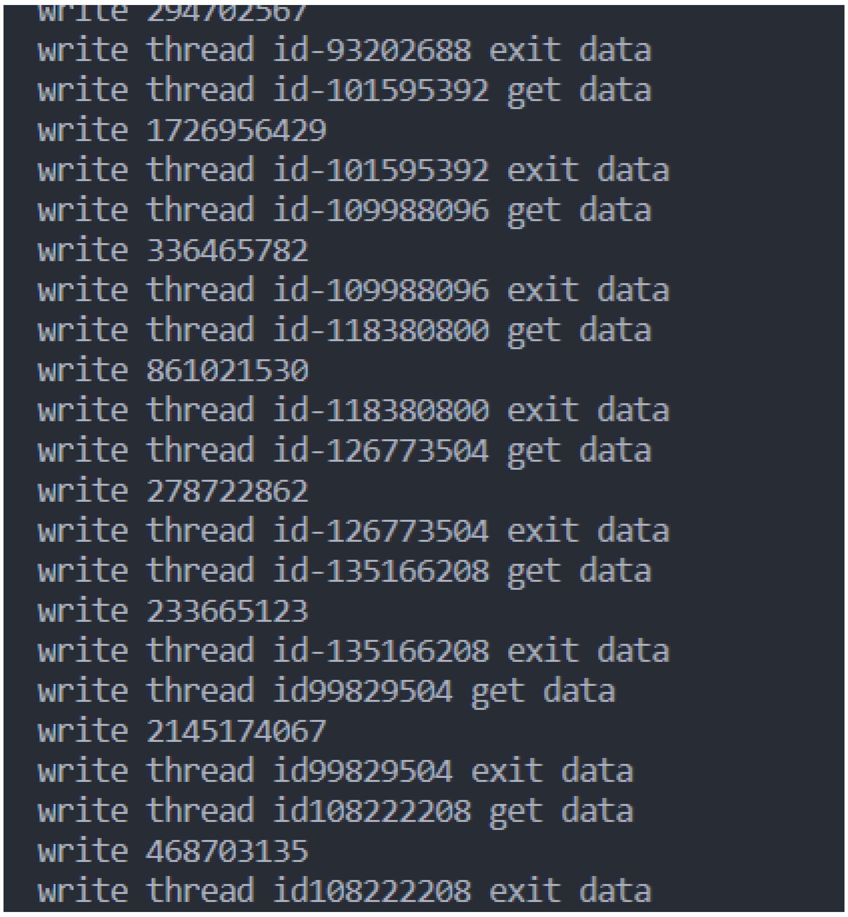


```
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ touch thread1.c
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ gcc thread1.c -o thread1 -lpthread
thread1.c: In function 'writer':
thread1.c:32:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'pthread_t {aka long unsigned int}' [-Wformat=]
     printf("write thread id%d get data\n",pthread_self());
            ^
thread1.c:34:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'pthread_t {aka long unsigned int}' [-Wformat=]
     printf("write thread id%d exit data\n",pthread_self());
            ^
thread1.c:36:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
     sleep(W_SLEEP);
     ^
thread1.c: In function 'reader':
thread1.c:48:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'pthread_t {aka long unsigned int}' [-Wformat=]
     printf("read thread id%d get data\n",pthread_self());
            ^
thread1.c:50:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'pthread_t {aka long unsigned int}' [-Wformat=]
     printf("read thread id%d exit data\n",pthread_self());
            ^
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ 
```

运行程序



```
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ ./thread1
read first
write thread id57865984 get data
write 1804289383
write thread id57865984 exit data
write thread id49473280 get data
write 846930886
write thread id49473280 exit data
write thread id66258688 get data
write 1681692777
write thread id66258688 exit data
write thread id41080576 get data
write 1714636915
write thread id41080576 exit data
write thread id32687872 get data
write 1957747793
write thread id32687872 exit data
write thread id74651392 get data
write 424238335
write thread id74651392 exit data
write thread id24295168 get data
```



```
write 294702567
write thread id-93202688 exit data
write thread id-101595392 get data
write 1726956429
write thread id-101595392 exit data
write thread id-109988096 get data
write 336465782
write thread id-109988096 exit data
write thread id-118380800 get data
write 861021530
write thread id-118380800 exit data
write thread id-126773504 get data
write 278722862
write thread id-126773504 exit data
write thread id-135166208 get data
write 233665123
write thread id-135166208 exit data
write thread id99829504 get data
write 2145174067
write thread id99829504 exit data
write thread id108222208 get data
write 468703135
write thread id108222208 exit data
```

2. 写者优先

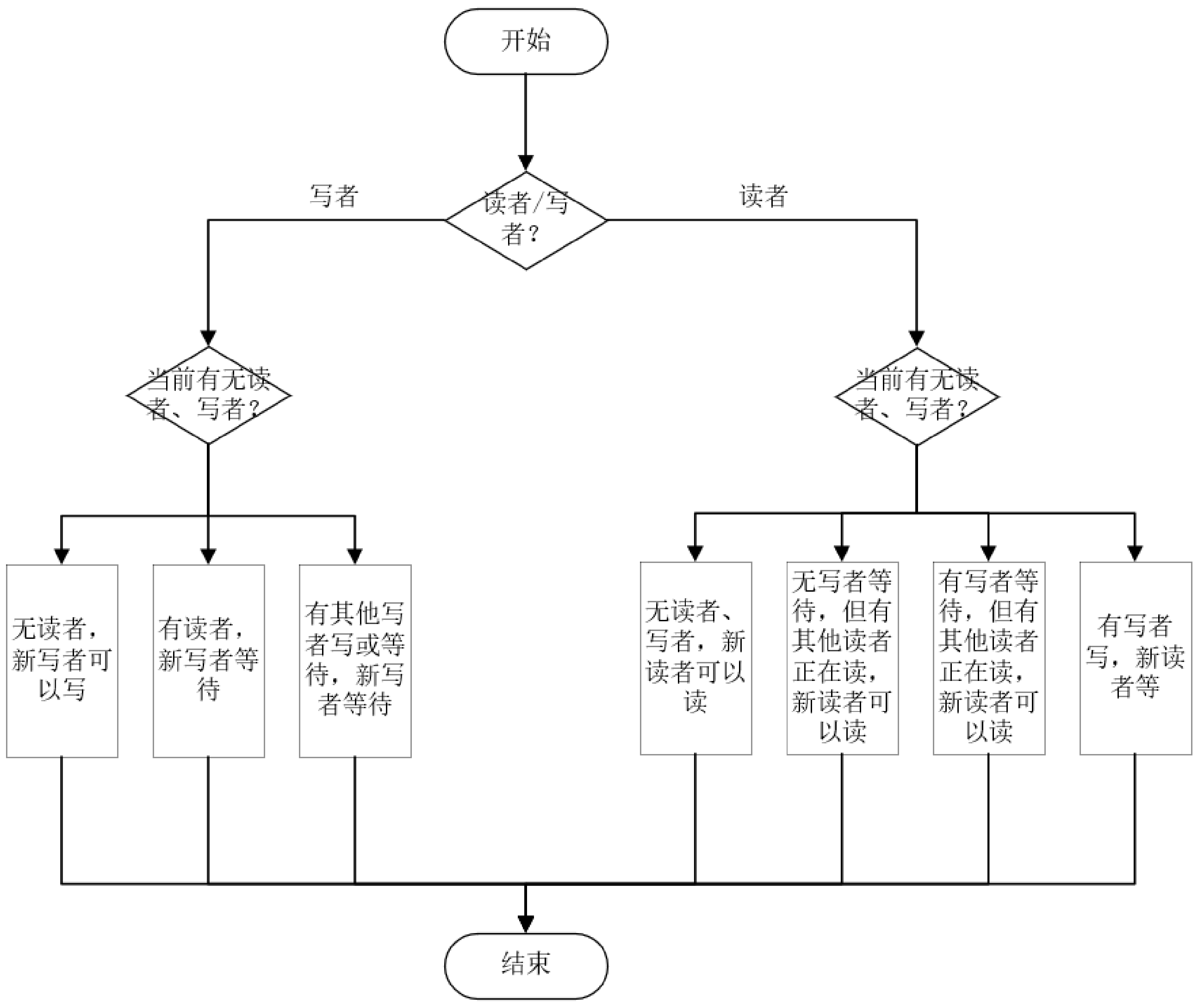
写者优先描述
如果读者来:

- 无读者、写者，新读者可以读；
- 无写者等待，但有其他读者正在读，新读者可以读；
- 有写者等待，但有其他读者正在读，新读者等；
- 有写者写，新读者等

如果写者来：

- 无读者，新写者可以写；
- 有读者，新写者等待；
- 有其他写者或等待，新写者等待

写者优先算法流程图如下



- thread2.c

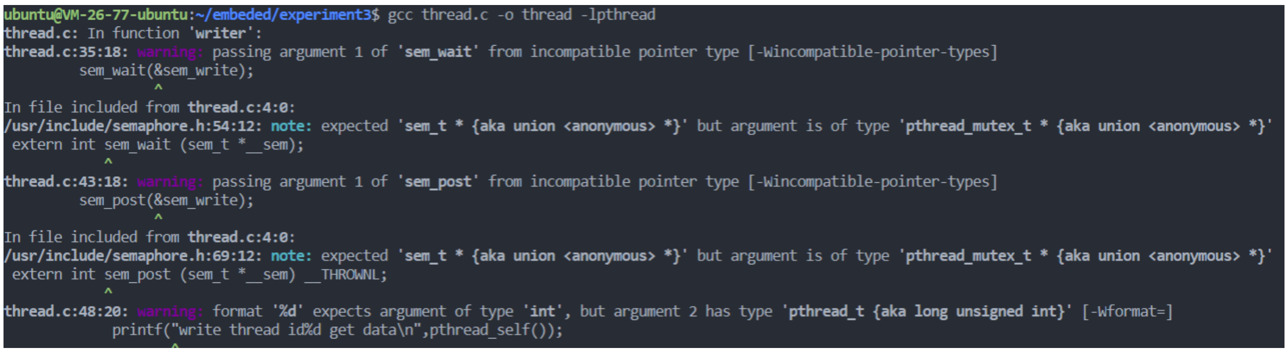
```
#include "stdio.h"
#include <stdlib.h>
#include <pthread.h>
#include<semaphore.h>
```

```
#define N_WRITER 5 //写者数目
#define N_READER 20 //读者数目
#define W_SLEEP 1 //控制写频率
#define R_SLEEP  0.5 //控制读频率

pthread_t wid[N_WRITER],rid[N_READER];
int data = 0;
int readerCnt = 0, writerCnt = 0;
pthread_mutex_t sem_read;
pthread_mutex_t sem_write;
pthread_mutex_t mutex_write;
pthread_mutex_t mutex_read;


void write()
{
    int rd = rand();
    printf("write %d\n",rd);
    data = rd;
}
void read()
{
    printf("read %d\n",data);
}
void * writer(void * in)
{

        sem_wait(&sem_write);
        {//临界区,希望修改 writerCnt,独占 writerCnt
            writerCnt++;
            if(writerCnt == 1){
                //阻止后续的读者加入待读队列
                pthread_mutex_lock(&mutex_read);
            }
        }
        sem_post(&sem_write);


        pthread_mutex_lock(&mutex_write);
        {//临界区，限制只有一个写者修改数据
            printf("write thread id%d get data\n",pthread_self());
            write();
            printf("write thread id%d exit data\n",pthread_self());
        }
        pthread_mutex_unlock(&mutex_write);

        sem_wait(&sem_write);
        {//临界区,希望修改 writerCnt,独占 writerCnt
            writerCnt--;
            if(writerCnt == 0){
                //阻止后续的读者加入待读队列
                pthread_mutex_unlock(&mutex_read);
            }
            }
            sem_post(&sem_write);
            sleep(W_SLEEP);
        pthread_exit((void *) 0);
    }


    void * reader (void * in)
    {
            //假如写者锁定了mutex_read,那么成千上万的读者被锁在这里
                pthread_mutex_lock(&mutex_read);//只被一个读者占有
                {//临界区
                    sem_wait(&sem_read);//代码段 1
                    {//临界区
                        readerCnt++;
                        if(readerCnt == 1){
                            pthread_mutex_lock(&mutex_write);
                        }
                    }
                    sem_post(&sem_read);
                }
                pthread_mutex_unlock(&mutex_read);//释放时,写者将优先获得mutex_read
            printf("read thread id%d get data\n",pthread_self());
            read();
            printf("read thread id%d exit data\n",pthread_self());
            sem_wait(&sem_read);//代码段2
            {//临界区
                readerCnt--;
                if(readerCnt == 0){
                    pthread_mutex_unlock(&mutex_write);//在最后一个并发读者读 完这里
开始禁止写者执行写操作
                }
            }
            sem_post(&sem_read);

            sleep(R_SLEEP);
        pthread_exit((void *) 0);
    }


    int main()
    {
        printf("write first\n");
        pthread_mutex_init(&mutex_write,NULL);
        pthread_mutex_init(&mutex_read,NULL);
        sem_init(&sem_write,0,1);
        sem_init(&sem_read,0,1);
    int i = 0;
        for(i = 0; i < N_READER; i++)
        {
            pthread_create(&rid[i],NULL,reader,NULL);
        }
        for(i = 0; i < N_WRITER; i++)
        {
```

```c
            pthread_create(&wid[i],NULL,writer,NULL);
        }
        sleep(1);
        return 0;
    }
```

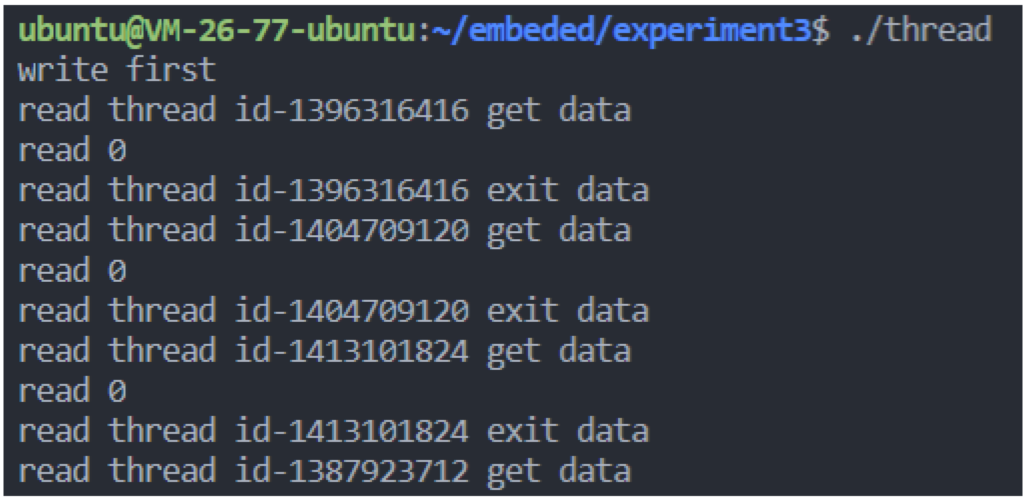
- 实验结果

  编译文件

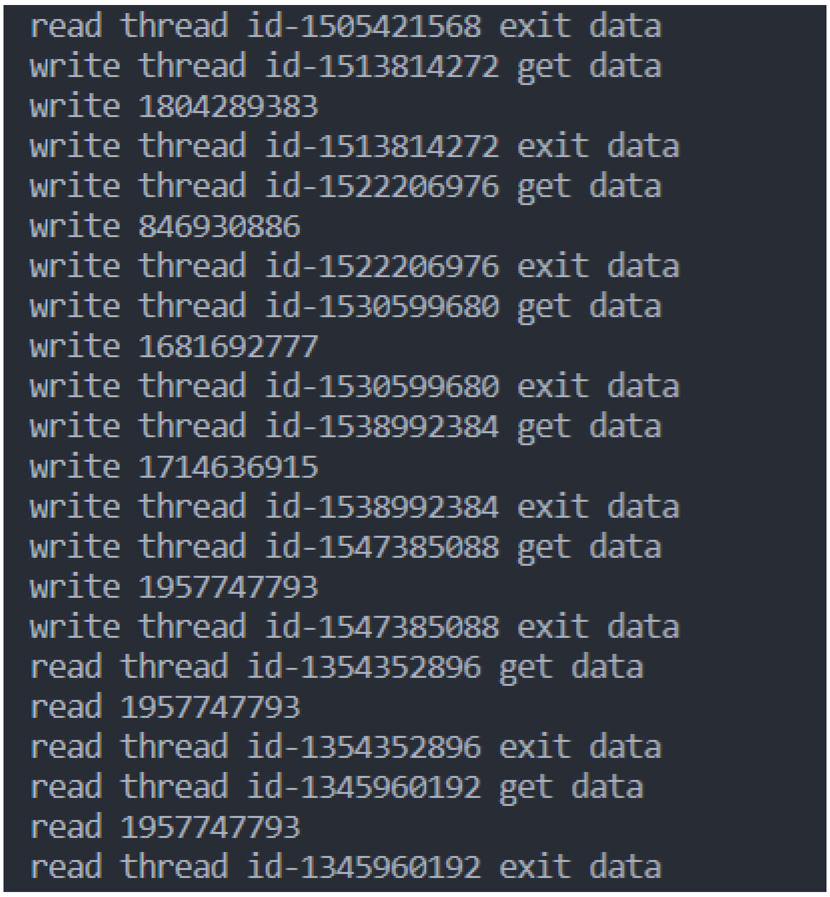


```
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ gcc thread.c -o thread -lpthread
thread.c: In function 'writer':
thread.c:35:18: warning: passing argument 1 of 'sem_wait' from incompatible pointer type [-Wincompatible-pointer-types]
         sem_wait(&sem_write);
                  ^
In file included from thread.c:4:0:
/usr/include/semaphore.h:54:12: note: expected 'sem_t * {aka union <anonymous> *}' but argument is of type 'pthread_mutex_t * {aka union <anonymous> *}'
 extern int sem_wait (sem_t *__sem);
            ^
thread.c:43:18: warning: passing argument 1 of 'sem_post' from incompatible pointer type [-Wincompatible-pointer-types]
         sem_post(&sem_write);
                  ^
In file included from thread.c:4:0:
/usr/include/semaphore.h:69:12: note: expected 'sem_t * {aka union <anonymous> *}' but argument is of type 'pthread_mutex_t * {aka union <anonymous> *}'
 extern int sem_post (sem_t *__sem) __THROWNL;
            ^
thread.c:48:20: warning: format '%d' expects argument of type 'int', but argument 2 has type 'pthread_t {aka long unsigned int}' [-Wformat=]
             printf("write thread id%d get data\n",pthread_self());
                    ^
```

  运行程序



```
ubuntu@VM-26-77-ubuntu:~/embeded/experiment3$ ./thread
write first
read thread id-1396316416 get data
read 0
read thread id-1396316416 exit data
read thread id-1404709120 get data
read 0
read thread id-1404709120 exit data
read thread id-1413101824 get data
read 0
read thread id-1413101824 exit data
read thread id-1387923712 get data
```



```
read thread id-1505421568 exit data
write thread id-1513814272 get data
write 1804289383
write thread id-1513814272 exit data
write thread id-1522206976 get data
write 846930886
write thread id-1522206976 exit data
write thread id-1530599680 get data
write 1681692777
write thread id-1530599680 exit data
write thread id-1538992384 get data
write 1714636915
write thread id-1538992384 exit data
write thread id-1547385088 get data
write 1957747793
write thread id-1547385088 exit data
read thread id-1354352896 get data
read 1957747793
read thread id-1354352896 exit data
read thread id-1345960192 get data
read 1957747793
read thread id-1345960192 exit data
```

四、实验总结

本次实验我利用课程中学到的pthread线程库中的多线程和信号量等相关函数，实现了读写者问题的读者优先和写者优先算法，加深了对多线程编程的理解。