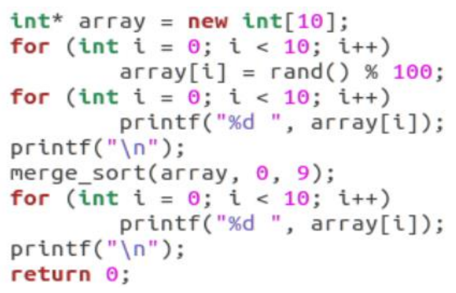
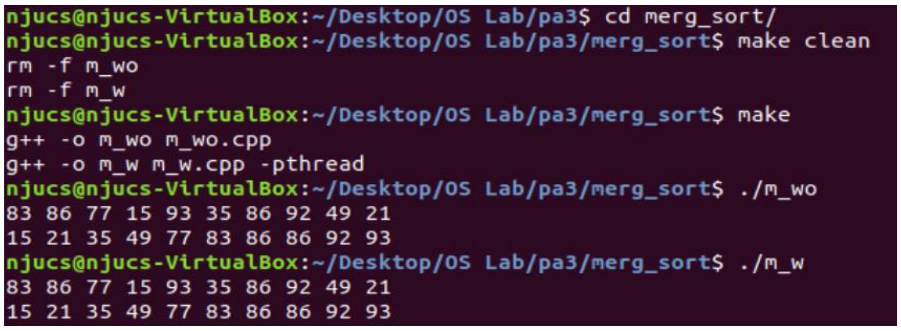
姓名：张涵之　　学号：191220154　　邮箱：1683762615@qq.com

➢ 算法并行化问题：merge_sort
非多线程版本：merge_sort/m_wo.cpp
简单的多线程版本-串行合并：merge_sort/m_w.cpp
调用系统函数 rand()生成伪随机数进行测试：



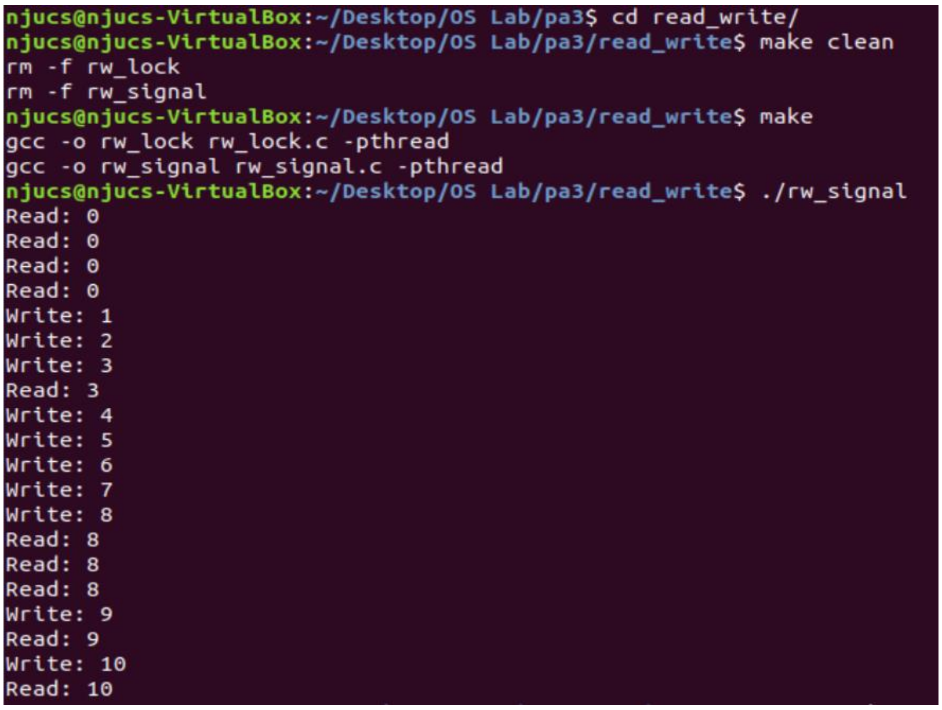
```
int* array = new int[10];
for (int i = 0; i < 10; i++)
        array[i] = rand() % 100;
for (int i = 0; i < 10; i++)
        printf("%d ", array[i]);
printf("\n");
merge_sort(array, 0, 9);
for (int i = 0; i < 10; i++)
        printf("%d ", array[i]);
printf("\n");
return 0;
```



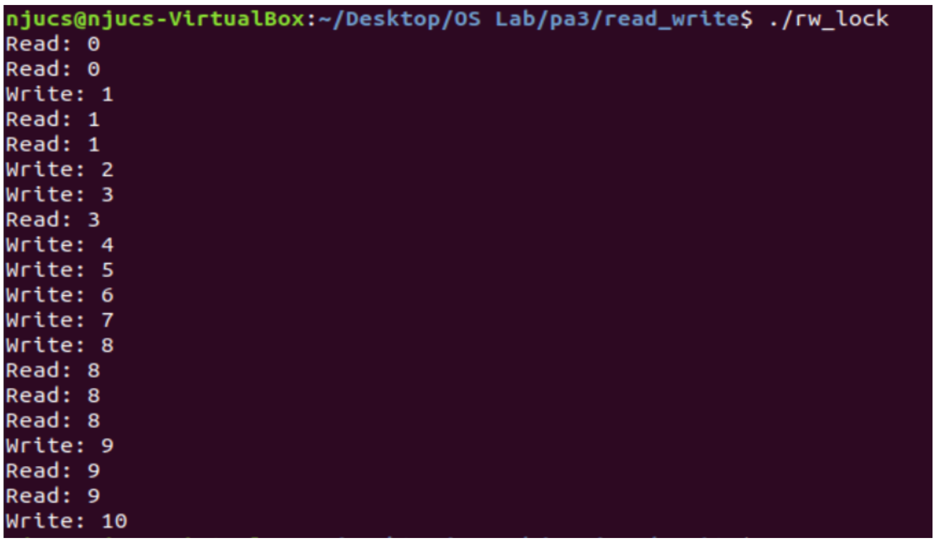
```
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3$ cd merg_sort/
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/merg_sort$ make clean
rm -f m_wo
rm -f m_w
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/merg_sort$ make
g++ -o m_wo m_wo.cpp
g++ -o m_w m_w.cpp -pthread
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/merg_sort$ ./m_wo
83 86 77 15 93 35 86 92 49 21
15 21 35 49 77 83 86 86 92 93
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/merg_sort$ ./m_w
83 86 77 15 93 35 86 92 49 21
15 21 35 49 77 83 86 86 92 93
```

并行版本的 P-MERGE 由于时间紧张，未能很好地理解，因此没有尝试。

➢ 信号量与 PV 操作实现同步问题：read_write
使用 pthread 提供的信号量与 PV 操作：rw_signal.c
使用 pthread 提供的读者-写者锁：rw_lock.c
设置全局变量 count 来模拟读写，检测是否冲突：



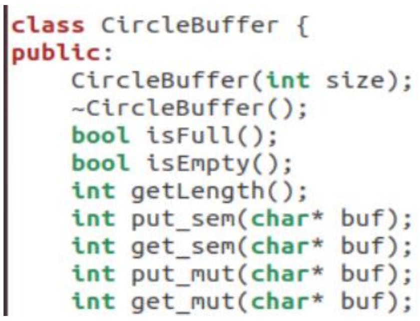
```
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3$ cd read_write/
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/read_write$ make clean
rm -f rw_lock
rm -f rw_signal
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/read_write$ make
gcc -o rw_lock rw_lock.c -pthread
gcc -o rw_signal rw_signal.c -pthread
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/read_write$ ./rw_signal
Read: 0
Read: 0
Read: 0
Read: 0
Write: 1
Write: 2
Write: 3
Read: 3
Write: 4
Write: 5
Write: 6
Write: 7
Write: 8
Read: 8
Read: 8
Read: 8
Write: 9
Read: 9
Write: 10
Read: 10
```



```
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/read_write$ ./rw_lock
Read: 0
Read: 0
Write: 1
Read: 1
Read: 1
Write: 2
Write: 3
Read: 3
Write: 4
Write: 5
Write: 6
Write: 7
Write: 8
Read: 8
Read: 8
Read: 8
Write: 9
Read: 9
Read: 9
Write: 10
```

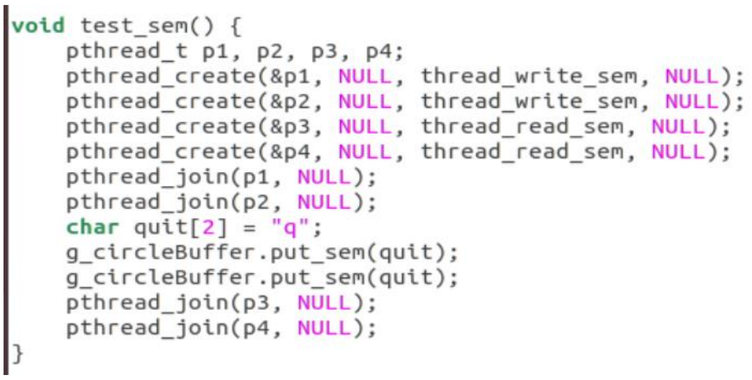
如图可见，两种方法都实现了正确的无冲突读写操作，效果看起来差不多。

- 使用 C++语言和 pthread 库实现一个有界环形缓冲区类(CircleBuffer)：
  使用 pthread 库提供的一般信号量(semaphore)：
  使用 pthread 库提供的互斥信号量(mutex)和条件变量(condition)：
  circle_buffer/c_buff.h & c_buff.cpp & main.cpp
  实际上我只写了一个 class，为两种实现提供不同的成员函数：

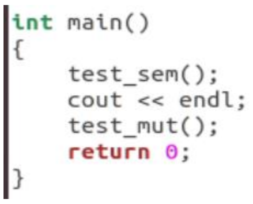


```
class CircleBuffer {
public:
    CircleBuffer(int size);
    ~CircleBuffer();
    bool isFull();
    bool isEmpty();
    int getLength();
    int put_sem(char* buf);
    int get_sem(char* buf);
    int put_mut(char* buf);
    int get_mut(char* buf);
```

在 main.cpp 中定义和调用不同的测试函数（以 test_sem 为例）：
其中 thread_read/write_sem 和 read/write_mut 分别调用对应的成员函数

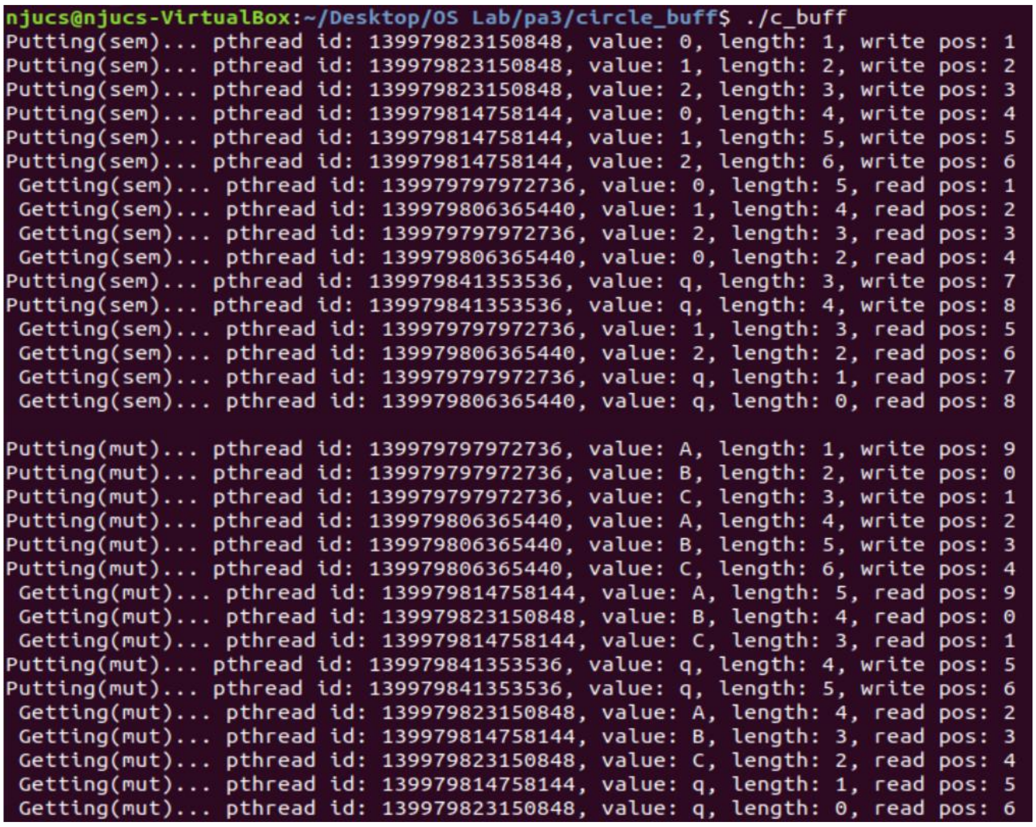


```
void test_sem() {
    pthread_t p1, p2, p3, p4;
    pthread_create(&p1, NULL, thread_write_sem, NULL);
    pthread_create(&p2, NULL, thread_write_sem, NULL);
    pthread_create(&p3, NULL, thread_read_sem, NULL);
    pthread_create(&p4, NULL, thread_read_sem, NULL);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    char quit[2] = "q";
    g_circleBuffer.put_sem(quit);
    g_circleBuffer.put_sem(quit);
    pthread_join(p3, NULL);
    pthread_join(p4, NULL);
}
```



```
int main()
{
    test_sem();
    cout << endl;
    test_mut();
    return 0;
}
```

运行截图如下，可见读写操作无冲突地实现了，类成员变量都可以正确更新：



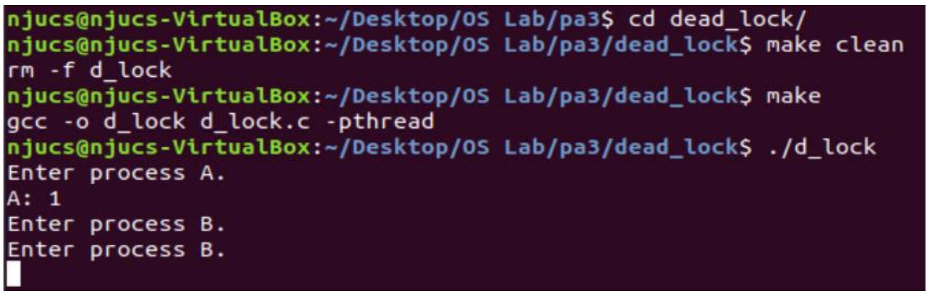
```
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/circle_buff$ ./c_buff
Putting(sem)... pthread id: 139979823150848, value: 0, length: 1, write pos: 1
Putting(sem)... pthread id: 139979823150848, value: 1, length: 2, write pos: 2
Putting(sem)... pthread id: 139979823150848, value: 2, length: 3, write pos: 3
Putting(sem)... pthread id: 139979814758144, value: 0, length: 4, write pos: 4
Putting(sem)... pthread id: 139979814758144, value: 1, length: 5, write pos: 5
Putting(sem)... pthread id: 139979814758144, value: 2, length: 6, write pos: 6
 Getting(sem)... pthread id: 139979797972736, value: 0, length: 5, read pos: 1
 Getting(sem)... pthread id: 139979806365440, value: 1, length: 4, read pos: 2
 Getting(sem)... pthread id: 139979797972736, value: 2, length: 3, read pos: 3
 Getting(sem)... pthread id: 139979806365440, value: 0, length: 2, read pos: 4
Putting(sem)... pthread id: 139979841353536, value: q, length: 3, write pos: 7
Putting(sem)... pthread id: 139979841353536, value: q, length: 4, write pos: 8
 Getting(sem)... pthread id: 139979797972736, value: 1, length: 3, read pos: 5
 Getting(sem)... pthread id: 139979806365440, value: 2, length: 2, read pos: 6
 Getting(sem)... pthread id: 139979797972736, value: q, length: 1, read pos: 7
 Getting(sem)... pthread id: 139979806365440, value: q, length: 0, read pos: 8

Putting(mut)... pthread id: 139979797972736, value: A, length: 1, write pos: 9
Putting(mut)... pthread id: 139979797972736, value: B, length: 2, write pos: 0
Putting(mut)... pthread id: 139979797972736, value: C, length: 3, write pos: 1
Putting(mut)... pthread id: 139979806365440, value: A, length: 4, write pos: 2
Putting(mut)... pthread id: 139979806365440, value: B, length: 5, write pos: 3
Putting(mut)... pthread id: 139979806365440, value: C, length: 6, write pos: 4
 Getting(mut)... pthread id: 139979814758144, value: A, length: 5, read pos: 9
 Getting(mut)... pthread id: 139979823150848, value: B, length: 4, read pos: 0
 Getting(mut)... pthread id: 139979814758144, value: C, length: 3, read pos: 1
Putting(mut)... pthread id: 139979841353536, value: q, length: 4, write pos: 5
Putting(mut)... pthread id: 139979841353536, value: q, length: 5, write pos: 6
 Getting(mut)... pthread id: 139979823150848, value: A, length: 4, read pos: 2
 Getting(mut)... pthread id: 139979814758144, value: B, length: 3, read pos: 3
 Getting(mut)... pthread id: 139979823150848, value: C, length: 2, read pos: 4
 Getting(mut)... pthread id: 139979814758144, value: q, length: 1, read pos: 5
 Getting(mut)... pthread id: 139979823150848, value: q, length: 0, read pos: 6
```

- 死锁问题：dead_lock

  实现一个多线程程序在运行中发生死锁：d_lock.c

  定义两个函数 A 和 B，在 A 的临界区内调用 B，加入并行的线程：



```
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3$ cd dead_lock/
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/dead_lock$ make clean
rm -f d_lock
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/dead_lock$ make
gcc -o d_lock d_lock.c -pthread
njucs@njucs-VirtualBox:~/Desktop/OS Lab/pa3/dead_lock$ ./d_lock
Enter process A.
A: 1
Enter process B.
Enter process B.
█
```

如图可见，进入线程 A 的临界区后，调用 B，同时单独的线程 B 也进入。

两个 B 中任何一个都无法进入临界区执行，则 A 无法退出临界区，形成死锁。