**线程同步与线程同步的必要性**

**线程同步的必要性**

所谓 线程同步(synchronization)：指的就是在一个线程访问数据未结束时，其他线程不能对同一个数据进行访问。  
如此，便可以保证对数据的访问是原子化的。

**没有线程同步时，发生的数据错乱**

//linux下 main.c 该代码没有线程同步措施，导致数据出错

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

int number = 0;

void\* counterA(void\* arg) {

for (int i = 0; i < 50; i++) {

int cur = number;

cur++;

usleep(4);

number = cur;

printf("线程A---线程A地址：%p---number:%d\n", pthread\_self(), number);

}

}

void\* counterB(void\* arg) {

for (int i = 0; i < 50; i++) {

int cur = number;

cur++;

usleep(4);

number = cur;

printf("线程B---线程B地址：%p---number:%d\n", pthread\_self(), number);

}

}

int main()

{

pthread\_t t1;

pthread\_t t2;

pthread\_create(&t1, NULL, counterA, NULL);

pthread\_create(&t2, NULL, counterB, NULL);

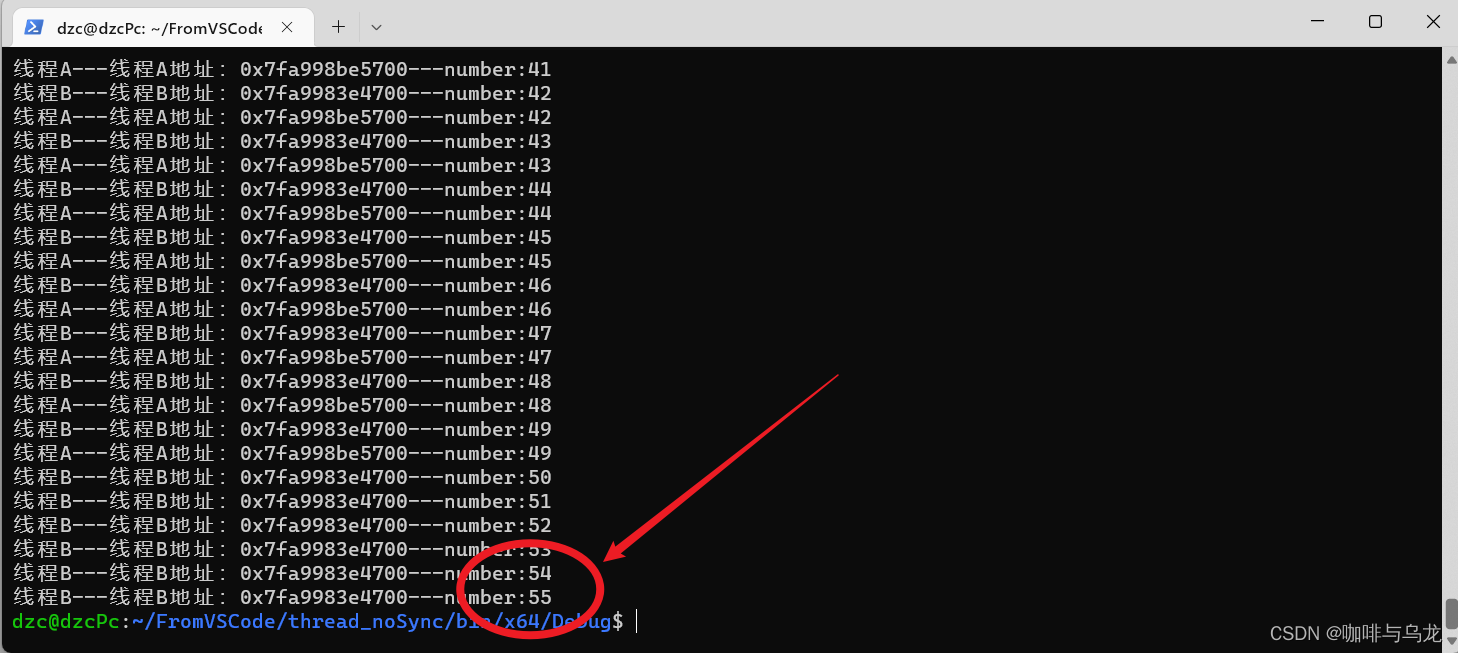
pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

return 0;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

程序运行结果如下所示：  


* 上述代码**没有加锁等线程同步机制**，并且两个线程（A、B）访问了一个共享资源（全局变量number），因此会发生竞争，最终导致数据错乱（按理说数数，应该数到100）

**原因：**

①某一个**线程A**获取共享资源是从内存读取到CPU进行处理，当CPU时间片用完后，未能及时将最新数据更新到内存中；

②当另一个**线程B**抢到CPU时间片开始工作，同样的获取共享资源是从内存中读取到CPU中，但是此时内存中的数据已经不是正确的最新数据了，因此发生错误;

③当**线程A**又一次获得时间片时，第一步就是把寄存器中的数据给更新到共享资源的内存中，因此就会覆盖刚刚线程B的工作结果，又导致错误。

**总结**：

因此**线程同步**就非常重要，常用的线程同步方式就是**加锁**。锁是一种非强制机制，每一个线程在访问数据或资源之前都应该获取锁（或者说上锁、lock），并在访问结束之后释放锁(或者说解锁、unlock)。在锁被别的线程占用时，当前线程就需要等待也即是阻塞，直到锁被释放，然后当前线程抢到锁了，才能继续执行。

参考文献：  
1、《程序员的自我修养》  
2、[苏丙榅大佬的博客](https://subingwen.cn/linux/thread-sync/)