一、书上 4.7

Many current language specifications, such as for C and C++, are inadequate for multithreaded programs. This can have an impact on compilers and the correctness of code, as this problem illustrates. Consider the following declarations and function definition:

```
int global_positives = 0;
typedef struct list {
   struct list *next;
   double val;
} * list;
void count_positives(list l)
{
   list p;
   for (p = l; p; p = p -> next)
   if (p -> val > 0.0)
   ++global_positives;
}
```

Now consider the case in which thread A performs count_positives(<list containing only negative values>); while thread B performs ++global_positives;

a. What does the function do?

统计链表 list 中正数的个数

b. The C language only addresses single-threaded execution. Does the use of two parallel threads create any problems or potential problems?

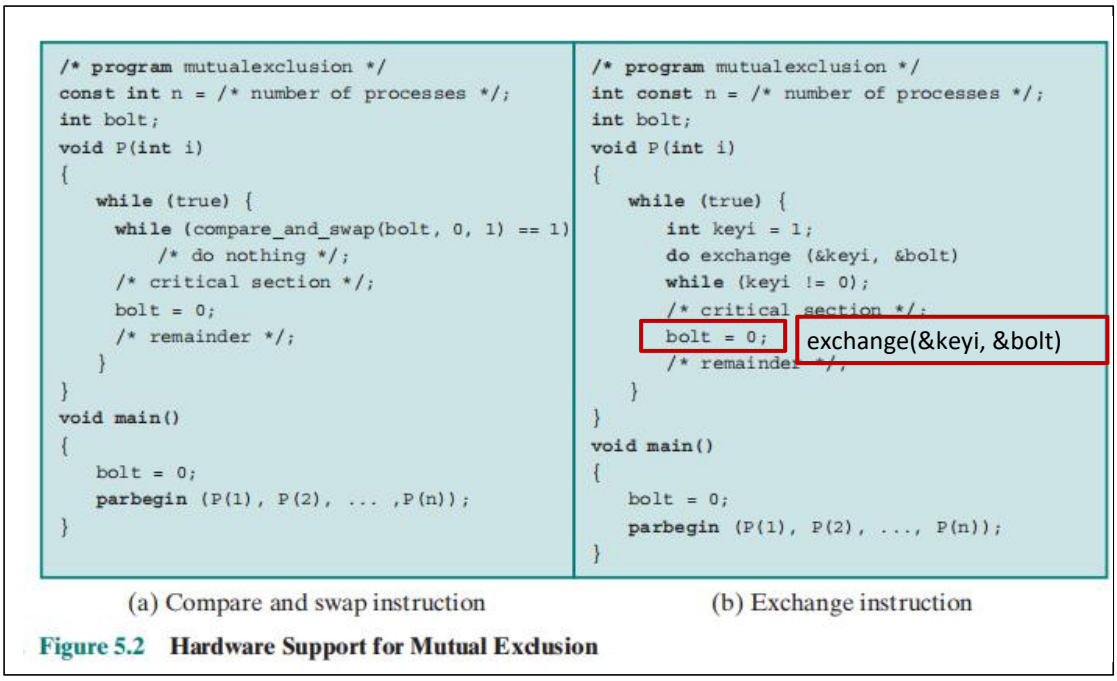
就本题中的情况来说，对全局变量 global_positives 的访问不会发生问题，因为线程 A 执行函数时，由于 list 中全为负数，不会访问 global_positives 变量，线程 B 的 global_positives 访问相当于是排他互斥的。但考虑更一般的情况，会发生问题，global_positives 是临界资源，需要互斥访问才行。

二、书上 163 页 5.10

Consider the first instance of the statement bolt = 0 in Figure 5.2b.

a. Achieve the same result using the exchange instruction.

b. Which method is preferable?



(a) Compare and swap instruction (b) Exchange instruction

Figure 5.2 Hardware Support for Mutual Exclusion

a. exchange (&keyi, &bolt)，之前 keyi 的值是 0，bolt 的值是 1，交换后 bolt 的值被置为 0，keyi 的值被置为 1，可实现与 bolt=0 语句相同的功能。

b. 赋值语句与原子操作（硬指令）的区别，从执行效率等方面考虑，用 exchange 好一点

以下文字为学生答案

b.用exchange更好,用exchange能保证任何时间keyi与bolt的值的总和都为n,若直接令bolt=0,这个刚释放临界资源的进程此时的keyi还是0,另一个进程进入临界区后keyi也为0,同一时间有两个进程的keyi为0,无法精准定位哪个进程在临界区中,采用exchange就避免了这种情况,离开临界区后keyi就被换回了1.任何时间只有keyi=0的进程在临界区中.

从并发控制看，即使用赋值语句，也不影响临界区访问的正确性，但会存在如上的情况。

三、书上 163 页 5.11

When a special machine instruction is used to provide mutual exclusion in the fashion of Figure 5.2, there is no control over how long a process must wait before being granted access to its critical section. Devise an algorithm that uses the compare&swap instruction but that guarantees that any process waiting to enter its critical section will do so within n －1 turns, where n is the number of processes that may require access to the critical section and a “turn” is an event consisting of one process leaving the critical section and another process being granted access.

int lock;//锁

int waiting[n];//状态数组，进程i是否在进入区等待进入

```
void P(int i)
{    var    int j;
            Int key;
       while (true) {
```

```
            waiting[i] =1;//进程i等待进入
            key = 1;
            while (waiting[i] && key)
                key =compare_and_swap(lock, 0, 1) ;//key为返回的锁状态，为0
时进入临界区
            waiting[i] = 0;//已进入，不再等待
            < critical section >
            j = (i + 1) % n;//从进程i+1起扫描一圈到进程i-1,最多n-1个进程想进
        入
            while (j != i && !waiting[j])
                j = (j + 1)% n;//找到第一个在进入区等待进入的进程（waiting[j]
为1的）
            if (j == i)
                lock = 0//一圈后没有找到进程等待进入，释放锁
            else waiting[j] = 0;//否则，让第一个这样的进程进入临界区
        （ waiting[j] 置为0，while (waiting[i] && key)不成立，进入临界区），最
        坏n-1轮后进入临界区
            < remainder section >
        }
}
void main()
{
    lock=0;
    waiting[n]={0}
    parbegin(P(0),P(1),...,P(n-1));
}
```

四、165 页 5.14

1、能保证最多三个进入 CS；不到三个时，申请的进程可立即进入；已有三个时，申请进程等待；只有三个进程都退出时，再允许最多三个进入 CS。阻塞在信号量 block 上的进程被唤醒后，没有互斥信号量 mutex 的 2 次申请；waiting、active 计数器值的更新由唤醒者完成，且更新后，再唤醒对应的阻塞在信号量 block 上的进程，使得计数器值能准确反映出当前的系统状态（对比 5.13 来看）

2 、CS 已有三个进程，又来了 p4、p5、p6，waiting 值加到了 3，但 p6 未执行到第 9 行代码（mutex 已释放，block=-2），若进入 CS 的前三个进程依次离开，最后一个更新计数器 waiting=0，active=3，执行三次 semsignal(block)（p4、p5 进入 CS，block=1）,must_wait=true 后，新来进程 P7 执行第 9 行后（block=0），可进入 CS，而此时 p6 再执行第 9 行，被阻塞（block=-1）

3、根据所唤醒进程的个数等，由唤醒者为被唤醒者更新计数器（系统状态）

五、三个并发进程 R、W1、W2 共享单缓冲区 B。进程 R 不断从输入设备上读入一个自然数存放到 B 中，若 B 中的数是奇数，则由进程 W1 取出打印，若 B 中的数是偶数，则由进程 W2 取出打印。用管程机制实现并发控制问题，使这三个进程能正确执行

```
moniter    bufferOE;
cond notfull,odd,even;      //条件变量，分别表示缓冲区不满，奇数，偶数
enum {0，1，2} flag;  //定义枚举类型变量 flag,分别表示缓冲区空，有奇数，有偶数，
int B；
Append(x):
{
         If ( flag!=0) cwait(notfull); //缓冲区满阻塞
        B=x;
         if(x%2!=0)   {flag=1;csignal(odd);} //放了奇数
         else    {flag=2;csignal(even);} //放了偶数
 }
Take1(x):       //取奇数
{
       If (flag!=1 ) cwait(odd);
       x=B;
       flag=0；
       csignal(notfull);

}
Take2(x):          //取偶数
{
        If (flag!=2 ) cwait(even);
         x=B;
       flag=0；
       csignal(notfull);
 }
{ flag=0；}//缓冲区状态标志初始为空

void R()
{int x;
  while(true)
```

```
    {
        Input(x);
        bufferOE.Append(x);
    }
}
void W1()
{int x;
  while(true)
    {
        bufferOE.Take1(x);
        print(x);
    }
}
void W2()
{int x;
  while(true)
    {
        bufferOE.Take2(x);
        print(x);
    }
}
void main()
{
     parbegin(R,W1,W2);
}
```

六、用消息机制解决司机、售票员问题。

司机进程：

```
message msgd;
while(true){
   recieve(box1, msgd);
   启动车辆;
   正常驾驶;
   到站停车;
   send(box2, msgd);
}
```

售票员进程：

```
message msgc=null;
while(true){
   关门;
   send(box1, msgc);
   售票;
   recieve(box2, msgc);
```

开门;
}
void main(){
create mailbox(box1);
create mailbox(box2);
parbegin(司机进程，售票员进程);
}

七

```
/* program boundedbuffer */
      const int sizeofbuffer = /* buffer size */;
      semaphore s = 1, n = 0, e = sizeofbuffer;
      void producer()
      {
           while (true) {
                produce();
                semWait(e);
                semWait(s);
                append();
                semSignal(s);
                semSignal(n);
           }
      }
      void consumer()
      {
           while (true) {
                semWait(n);
                semWait(s);
                take();
                semSignal(s);
                semSignal(e);
                consume();
           }
      }
      void main()
      {
                parbegin (producer, consumer);
      }
```

考虑上图，交换以下每对原语的先后次序，程序的含义是否改变?
a. semWait(e);semWait(s)
b. semSignal(s);semSignal(n)
程序的含义改变了，但可能的后果不同
a. 交换后，“semWait(e);”语句成为临界区的一部分，程序含义改变，使得临界区释放滞后，在特定的执行顺序下，可能导致死锁:
假设当前缓冲区空（e=0），临界区无进程（s=1），生产进程执行 semWait(s)成功（s=0），执行 semWait(e)阻塞（e=-1）；消费进程执行 semWait(n)成功，但执行 semWait(s)阻塞。如果是多个生产进程、消费进程，那么第一个生产进程阻塞在 e 上，后继生产进程会依次阻塞在 s 上；前 buffersize 个消费进程会阻塞

在 s 上（n 递减至 0），再来的进程会阻塞在 n 上。系统出现死锁。

b. 交换后，“semSignal(n);”语句成为临界区的一部分，程序含义改变，使得临界区释放滞后，但不影响并发进程的执行