第六题（10 分）

1. 火锅洞洞主终于要请吃火锅啦！由于防疫规定，洞主决定分批邀请，Alice、Bob、……、Zach 这 26 位同学顺理成章地成为了第一批受到邀请的同学。**他们围坐在一张圆桌旁，按照首字母顺序排列**（即 Alice 的右边是 Bob，Bob 的右边是 Carol，……，Zach 的右边是 Alice）。吃饭之前，洞主想要验证他们的身份，他提出了这样的要求：

   1）每位同学有一个**编号**，即**字母序号减 1**，也就是说，Alice 是 0 号，Bob 是 1 号，以此类推。

   2）每位同学需要同时拿着**旁边两位同学**的手机去找洞主（**不必带上自己的手机**），按照洞主的要求在树洞里发信息以验证身份。

   3）**可以同时有多位同学**找洞主验证（也就是说，Alice 需要拿着 Zach 和 Bob 的手机找洞主，并且在同一时间，Bob 也可以拿着 Alice 和 Carol 的手机找洞主。但是，如果 Alice 还没同时拿到 Zach 和 Bob 的手机，就**只能坐在座位上等待**。）

所幸，在座有多位字母君学过 ICS，他们决定采用**信号量**解决问题。

1.1 Alice 给出了这样的代码，其中 thread 函数的参数指明了字母君的编号：

```
1.  #include "csapp.h"
2.  #define N 26
3.  #define UP(i) ((i + 1) % N)
4.  #define DOWN(i) ((i - 1 + N) % N)
5.  sem_t sem[N];
6.
7.  void *thread(void *i){
8.      int num = (int)i;
9.      P(&sem[UP(num)]);
10.     P(&sem[DOWN(num)]);
11.
12.     /* verify identity */
13.
14.     V(&sem[UP(num)]);
15.     V(&sem[DOWN(num)]);
16. }
17. int main(){
18.     for (int i = 0; i < N; i++)
19.         Sem_init(&sem[i], 0, 1);
20.     pthread_t tid[N];
21.     for (int i = 0; i < N; i++)
22.         Pthread_create(&tid[i], NULL, thread, (void *)i);
23.     Pthread_exit(0);
24. }
```

Bob 一看，皱起了眉头。请问这段代码可能会发生什么问题（用一个专用名词表述即可）？ 死锁

1.2 Carol 把 thread 函数换成了下面这样：

```
1.  void *thread(void *i){
2.      int num = (int)i;
3.      for (int i = 0; i < N; i++)
4.          P(&sem[i]);
5.      /* verify identity */
6.      for (int i = 0; i < N; i++)
7.          V(&sem[i]);
8.  }
```

等所有人都放下，每一时刻至多 1 人拿手机验证

Dave 指出，这段代码固然正确，但是效率太低了，他认为只要把第 3、4 行的代码换成一个 P 操作，把第 6、7 行的代码换成一个 V 操作，就能实现同样的效果。请问 Dave 的方法是：

1.2.1 P 操作：P(&sem[0]);

1.2.2 V 操作：V(&sem[0]);

1.3 Eve 看了 Dave 的方案还是摇头，这样执行效率太低了，没能利用**同时可以有多位**同学找洞主验证的条件。借助“**互斥锁加锁顺序规则**”，他给出了自己的代码。你能把代码补全吗？

```
1.  void *thread(void *i)
2.  {
3.      int num = (int)i;
4.      if (UP(num) < DOWN(num)){
5.          P(&sem[ A ]);
6.          P(&sem[ B ]);
7.      }
8.      else{
9.          P(&sem[ C ]);
10.         P(&sem[ D ]);
11.     }
12.     /* verify identity */
13.     V(&sem[UP(num)]);
14.     V(&sem[DOWN(num)]);
15. }
```

up < down

若都选大的先拿，为 0 和 1 一开始这人拿

1 若能拿到 2，为 1 拿 0 即可（就到说明 25 拿了 24 和 0）

1 若到 2，说明 1 拿了 0 和 2

若都选小的先拿，为 25 和 24 一开始没人拿

24 若能拿到 23，为 24 拿 25 即可（就到为 23 拿了 22 和 24）

24 若到 23，为 22 拿了 21 和 23

不会死锁

A: ② B: ① C: ① D: ②

以上 4 空均填序号：

① UP(num)

② DOWN(num)

③ num

2. Hungary Alice 发现自己没吃上第一场的火锅，一怒之下，给上面的 26 位字母君出了一道难题：**用二元信号量实现信号量**（也就是值可以是任意非负整数的信号量）。精通 ICS 的你自然认为这是小儿科，迅速补全了以下代码。

```
1.  #include "csapp.h"
2.  typedef struct{
3.      int value;
4.      sem_t mutex;
5.      sem_t zero;
6.  } my_sem_t;
7.  void my_sem_init(my_sem_t *sem, unsigned int value){
8.      sem->value = value;
9.      Sem_init(&sem->mutex, 0, __A__);
10.     Sem_init(&sem->zero, 0, __B__);
11. }
12. void my_P(my_sem_t *sem){
13.     P(&sem->mutex);
14.     sem->value--;
15.     if (sem->value __C__ 0)
16.     {
17.         __D__;
18.         __E__;
19.     }
20.     else
21.         V(&sem->mutex);
22. }
23. void my_V(my_sem_t *sem){
24.     P(&sem->mutex);
25.     sem->value++;
26.     if (sem->value __F__ 0)
27.     {
28.         V(&sem->mutex);
29.         V(&sem->zero);
30.     }
31.     else
32.         V(&sem->mutex);
33. }
```

A: 1　　B: 0　　C: <

D: ③　　E: ②　　F: <=

D 和 E 请填序号：

① P(&sem->mutex)

② P(&sem->zero)

③ V(&sem->mutex)

④ V(&sem->zero)

1. 死锁。可以看出，这实际上就是哲学家就餐问题，如果所有的同学同时拿起左边那个同学手机 `P(UP(i))`，就发生死锁。

2. 分别填 `0` 和 `0`。观察代码知道 Dave 的方法每次只有一位同学去找洞主，也就是说 26 个同学轮流去验证。由此知道只需要一个互斥锁即可。

3. ①②②①。根据 1 问的例子可以知道依赖关系成环是最大的问题。因此我们可以让除一位同学之外的同学正常工作，而那位同学则反过来工作，不难证明这个体系无死锁且无竞争（事实上，还有一种方法，即要求单号同学先拿左边的手机，而双号先拿右边的手机）。

4. 分别填 `1`、`0`、`<`、③、②、`<=`。根据代码容易看出 `mutex` 保护结构体，`zero` 是一个“等待信号量”，所以计数降到 0 以下时需要等待，由此发现其初值应该是 0。请注意 ③、② 不能反，否则会造成死锁。

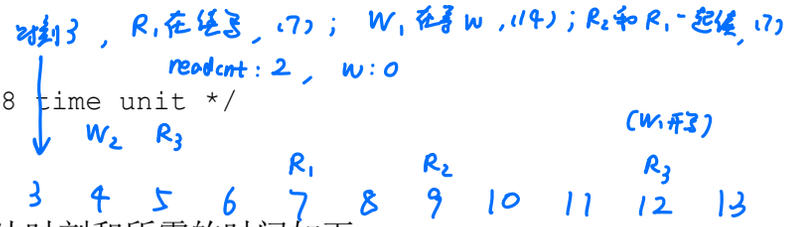
得分

第七题（10 分）

解决第一类读者-写者问题的代码如下：

```
int readcnt;    /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */
void reader(void)
{
(1)    while (1) {
(2)      P(&mutex);
(3)      readcnt++;
(4)      if (readcnt == 1) /* First in */
(5)        P(&w);
(6)      V(&mutex);
(7)         /* Reading happens here, need 7 time unit */
(8)      P(&mutex);
(9)      readcnt--;
(10)     if (readcnt == 0) /* Last out */
(11)       V(&w);
(12)     V(&mutex);
       }
}

void writer(void)
{
(13)  while (1) {
(14)    P(&w);
(15)       /* Writing here, need 8 time unit */
(16)    V(&w);
      }
}
```



假设有 5 个读者或写者到来，到达时刻和所需的时间如下：

线程	到达时刻	读写时间
R1	0	7
W1	1	8
R2	2	7
W2	4	8
R3	5	7

W权限：0~7 R1

2读 R2到，4读 R3到，5 R1一起读

注意，为简单起见，只考虑读写时间，忽略所有其他时间（包括代码中其他语句的执行时间、线程切换、调度等等），亦假设没有更多的读者或写者或其他线程。

（1）在时刻 3 时，R1、W1 和 R2 所处的位置，请标出对应的代码行号。此刻，readcnt 和 w 的值分别是多少？R1:7； W1:14； R2:7； readcnt:2； w:0。

（2）在时刻 6 时，W2 和 R3 所处的位置，请标出对应的行号。此刻，readcnt 和 w 的值分别是多少？W2:14； R3:7； readcnt:3； w:0。

（3）如果不使用 P(&mutex)和 V(&mutex)，程序执行会不会出错？如果出错，会出什么错？

时刻6，例定R1在读，W在R1等上

W2等w，(4)，R3一起读，(7)

readcnt：3 w：0

(3) 如果不用 P(&mutex)和 V(&mutex)，

会有竞争错误，比如两个读 Ri和Rk，一个刚读完在 readcnt--，一个刚进入在 readcnt++，

本来 readcnt 应1，但变 load 1，一个 store 0，一个store 2

错误地 store 了0，对开放了w权限，错误！

或 2个同时到来，readcnt 计数出错

1. `7, 14, 7, 2, 0`。这时两个读者都在正常读，因为是读者优先，所以写者在 14 行阻塞等待，写锁的值是 `0`。

2. `14, 7, 3, 0`。完全同理。

3. 会出现竞争的错误。例如两位读者同时进入，由于 `readcnt` 的写不是原子的，可能导致进入后 `readcnt=1`，当一位读者离开后，写者就错误进入了。