

## 读者写者算法-写者优先

```
1  // Reader
2  noReaders.wait()
3      readSwitch.lock(noWriters)
4  noReaders.signal()
5      //////////////////////////////////////////////////
6      // Critical Section for Readers
7      //////////////////////////////////////////////////
8      readSwitch.unlock(noWriters)
9
10 // Writer
11 writeSwitch.lock(noReaders)
12     noWriters.wait()
13         //////////////////////////////////////////////////
14         // Critical Section for Writers
15         //////////////////////////////////////////////////
16     noWriters.signal()
17 writeSwitch.unlock(noReaders)
18
19 // Config.
20 readSwitch = Lightswitch()
21 writeSwitch = Lightswitch()
22 noReaders = Semaphore(1)
23 noWriters = Semaphore(1)
```

最初的信号量都是解锁态。若Reader在临界区，会给noWriter上锁，但是不会给noReader上锁。如果这时候Writer到来，则会给noReader加锁，会让后续读者排队在noReader。当最后一个读者离开，他会signal noWriter，这时写者可以进入。

当写者进入临界区，则同时拥有noReader和noWriter两个锁。一方面，其他读者和写者不能同时访问临界区；另一方面，writeSwitch允许其他写者通过，并在noWriter等待，但是读者只能在noReader等待。以此所有的排队写者都可以通过临界区，而不需要signal noReader。当最后一个写者离开，noReader才解锁，写者才能进入。

## 寿司店问题

```
1  eating = 0
2  waiting = 0
3  mutex = Semaphore(1)
4  block = Semaphore(0)
```

```

5  must_wait = False
6
7  mutex.wait()
8  if must_wait:
9      waiting += 1
10     mutex.signal()
11     block.wait()
12 else:
13     eating += 1
14     must_wait = (eating == 5)
15     mutex.signal()
16
17 mutex.wait()
18 eating -= 1
19 if eating == 0:
20     n = min(5, waiting)
21     waiting -= n
22     eating += n
23     must_wait = (eating == 5)
24     block.signal(n)
25 mutex.signal()

```

## H2 三个进程P1、P2、P3互斥使用一个包含N (N>0) 个单元的缓冲区...

```

1  semaphore  mutex = 1;
2  semaphore  odd = 0;
3  semaphore  even = 1;
4  semaphore  empty = N;
5
6  P1() {
7      while(1) {
8          p(empty);
9          num = produce();
10         p(mutex);
11         put();
12         v(mutex);
13         if(num % 2 == 0)
14             v(even);
15         else
16             v(odd);
17     }
18 }
19

```

```

20 P2() {
21     while(1) {
22         p(odd);
23         p(mutex);
24         getodd();
25         countodd();
26         v(mutex);
27         v(empty);
28     }
29 }
30
31 P3() {
32     while(1) {
33         p(even);
34         p(mutex);
35         geteven();
36         counteven();
37         v(mutex);
38         v(empty);
39     }
40 }

```

## H2 搜索-插入-删除问题

```

1  # LightSwitch
2  insertMutex = Semaphore(1)
3  noSearcher = Semaphore(1)
4  noInserter = Semaphore(1)
5  searchSwitch = Lightswitch()
6  insertSwitch = Lightswitch()
7
8  # Searcher
9  searchSwitch.wait(noSearcher)
10 #####
11 # Critical Section
12 #####
13 searchSwitch.signal(noSearcher)
14
15 # Inserter
16 insertSwitch.wait(noInserter)
17 insertMutex.wait()
18 #####
19 # Critical Section
20 #####
21 insertMutex.signal()

```

```
22  insertMutex.signal(noInserter)
23
24  # Deleter
25  noSearcher.wait()
26  noInserter.wait()
27  #####
28  # Critical Section
29  #####
30  noInserter.signal()
31  noSearcher.signal()
32
33  # Config.
34  insertMutex = Semaphore(1)
35  noSearcher = Semaphore(1)
36  noInserter = Semaphore(1)
37
38  searcher = 0
39  searcherMutex = Semaphore(1)
40  inserter = 0
41  inserterMutex = Semaphore(1)
42
43  # Searcher
44  searcherMutex.wait()
45  searcher += 1
46  if searcher == 1:
47      noSearcher.wait()
48  searcherMutex.signal()
49
50  # Deleter
51  noSearcher.wait()
52  noInserter.wait()
53  #####
54  # Critical Section
55  #####
56  noInserter.signal()
57  noSearcher.signal()
58
59  #####
60  # Critical Section
61  #####
62
63  searcherMutex.wait()
64  searcher -= 1
65  if searcher == 0:
66      noSearcher.signal()
67  searcherMutex.signal()
68
69  # Inserter
70  inserterMutex.wait()
```

```
71  inserter += 1
72  if inserter == 1:
73      noInserter.wait()
74  inserterMutex.signal()
75
76  inserterMutex.wait()
77  #####
78  # Critical Section
79  #####
80  inserterMutex.signal()
81
82  inserterMutex.wait()
83  inserter -= 1
84  if inserter == 0:
85      noInserter.signal()
86  inserterMutex.signal()
```