1.

```
int readers = 0;           // 读者的数量
int writers = 0;           // 写者的数量
Semaphore readSignal = 1;     // 当有写进程将读进程堵塞
Semaphore mutexRead = 1;      // 保护对 readers 的访问
Semaphore mutexWrite =1;      // 保护对 writers 的访问
Semaphore wirtefile = 1;   // 保护写进程对临界区的访问
void Writer() {
   while(true) {
      P(mutexWrite);
      wirters = wirters + 1;
      if (writers == 1)
         P(readSignal);
      V(mutexWrite);
      P(writefile);
      wirte();
      V(writefile);
      P(mutexWrite);
      wirters = wirters - 1;
      if (writers == 0)
         V(readSignal);
      V(mutexWrite);
   }
}

void Reader() {
   while(true) {
      P(readSignal);
```

```
        P(mutexRead);
        readers=readers+1;
        if (readers == 1) //第一个读者
           P(roomEmpty);
        V(mutexRead);
        V(readSignal);
        read();         //critical region
        P(mutex);
        readers = readers-1;
        if (readers == 0)
           V(roomEmpty);
        V(mutex);
     }
  }
```

2.

```
  int seat = 0;           // 座位的数量
  int flag = 0;           // 是否满，flag=1 时为满
  Semaphore seatSignal = 1; // 保护对 seat 和 flag 的访问
  void Customer() {
     while(true) {
        P(full);
        P(seatSignal);
        seat = seat + 1;
        if (seat == 5){
           P(full);
           flag = 1;
```

```
        }
        V(seatSignal);
        V(full);
        seatAndEat();
        P(seatSignal);
        seat = seat - 1;
        if (flag == 1 && seat == 0) {
           P(full);
           flag = 0;
        }
        V(seatSignal);
     }
  }
```

3.

```
int searchCount = 0;       // 读进程的数量
Semaphore searchCountSignal = 1;
                           // 保护对 searchCount 的访问
Semaphore remvoeSignal = 1;      // 控制移除结点信号量
Semaphore insertSignal = 1;      // 控制插入结点信号量
Semaphore searchSignal = 1;      // 控制搜索结点信号量

void insert() {
   while(true) {
      P(removeSignal);
      P(insertSignal);
      operation_insert();
      V(insertSignal);
```

```
      V(removeSingal);
   }
}

void remove() {
   while(true) {
      P(removeSignal);
      P(insertSignal);
      P(searchSignal);
      operation_remove();
      V(searchSignal);
      V(insertSignal);
      V(removeSignal);
   }
}
void search() {
   while(true) {
      P(searchCountSignal);
      searchCount = searchCount + 1;
      if (searchCount > 0)
         P(searchSignal);
      V(searchCountSignal);
      operation_search();
      P(searchCountSignal);
      searchCount = searchCount - 1;
      if (searchCount == 0)
         V(searchSignal);
      V(searchCountSignal);
   }
}
```

4.

	已分配资源数	最大需求量	还需资源数	可用资源
进程 A	1 0 2 1 1	1 1 2 1 3	0 1 0 1 2	0 0 x 1 2
进程 B	2 0 1 1 0	2 2 2 1 0	0 2 1 0 0	
进程 C	1 1 0 1 0	2 1 3 1 0	1 0 3 0 0	
进程 D	1 1 1 1 0	1 1 2 2 1	0 0 1 1 1	

当 x = 0 时，A、B、C、D 四个进程都不能完成，陷入死锁

当 x = 1 时，D 进程可以完成，此时可用资源为 1 1 2 2 2，进程 A 可以完成，此时可用资源为 2 1 4 3 3，进程 C 可完成，此时可用资源为 3 2 4 4 3，进程 B 完成，所以 x 的最小值为 1