

```
1 #include "sem.h"
2 #include "spin.h"
3
4 static void dummy() //Dummy handler. Does nothing
5 {
6     // return;
7 }
8
9 void sem_init(struct sem *s, int count)
10 {
11     s->spinlock = 0;
12     s->max = count;
13     s->free = count;
14     s->procInd = -1;
15
16     sigfillset(&s->mask_block);
17     sigdelset(&s->mask_block, SIGUSR1);
18     signal(SIGUSR1, dummy); // Prevent the signal from killing the process
19 }
20
21 int sem_try(struct sem *s)
22 {
23     spin_lock(&s->spinlock);
24     if (s->free > 0)
25     {
26         s->free -= 1;
27         spin_unlock(&s->spinlock);
28         return 1;
29     }
30     else
31     {
32         spin_unlock(&s->spinlock);
33         return 0;
34     }
35 }
36
37 void sem_wait(struct sem *s)
38 {
39     for (;;)
40     {
41         spin_lock(&s->spinlock);
42
43         if (s->free > 0)
44         {
45             s->free -= 1;
46             spin_unlock(&s->spinlock);
47             break;
48         }
49         else
50         {
```

```
50     ^
51     s->proc_block[s->procInd] = procNum; // Put process on waitlist
52     s->procInd += 1; //Book keeping
53     spin_unlock(&s->spinlock);
54     sigsuspend(&s->mask_block); //Put process to sleep
55 }
56 }
57 }
58
59 void sem_inc(struct sem *s)
60 {
61     spin_lock(&s->spinlock);
62
63     s->free += 1; // Increment semaphore
64     if (s->free == 1)
65     {
66         while (s->procInd != -1) //Loop to wake up all processes when sem
67             becomes 1
68             {
69                 kill(pid_table[s->proc_block[s->procInd]], SIGUSR1);
70                 s->procInd -= 1; // Book keeping
71             }
72     }
73     spin_unlock(&s->spinlock);
74 }
```