VxWorks Task 之计数信号量

在<Task 之二进制信号量>里提到过二进制信号量用来解决同步问题。下面看一个同步的例子

```
#include <semLib.h> /* semBCreate */
#include <stdio.h>  /* printf */
#include <taskLib.h>/* taskName */

SEM_ID semId;

void testSemB()
    {
    semId=semBCreate(SEM_Q_FIFO, SEM_EMPTY);
    while(1)
        {
        semTake(semId,WAIT_FOREVER);

        printf("task %s took semaphore\n", taskName(0));
        }
    }
```

这段代码很简单，大致的意思就是每成功申请一次信号量，就打印一句话。
启动一个任务(t1)来调用这个函数:

```
-> sp testSemB
Task spawned: id = 0x38e11f0, name = t1
value = 59642352 = 0x38e11f0
-> w t1
  NAME       ENTRY       TID       STATUS    DELAY  OBJ_TYPE     OBJ_ID
---------- ---------- ---------- ---------- ----- ------------ ----------
t1         testSemB    0x38e11f0 PEND           0 SEM_B        0x1638b60
value = 0 = 0x0
-> show semId

Semaphore Id         : 0x1638b60
Semaphore Name       : N/A
Semaphore Type       : BINARY
Task Queuing         : FIFO
Pended Tasks         : 1
State                : EMPTY
Options              : 0x0       SEM_Q_FIFO
```

可以看到 t1 阻塞到信号量 semId 上了。直接给它释放一次信号量

```
-> semGive semId
value = 0 = 0x0
-> task t1 took semaphore
```

任务(t1)打印了一句话，说明收到了一次信号量
接下来试试释放两次信号量，可以用 Shell 的命令 repeat()

```
long repeat
    (
    int  n,       /* no. of times to call func (0=forever)*/
    FUNCPTR func,/* function to call repeatedly          */
    long arg1,    /* first of eight args to pass to func  */
    long arg2,
    long arg3,
    long arg4,
    long arg5,
    long arg6,
    long arg7,
    long arg8
    );
```

```
-> repeat 2,semGive,semId
Task spawned: id = 0x5993280, name = t2
value = 93926016 = 0x5993280
-> task t1 took semaphore
task t1 took semaphore
```

任务(t1)也打印了两句话，说明收到了两次信号量
接下来试试多次的

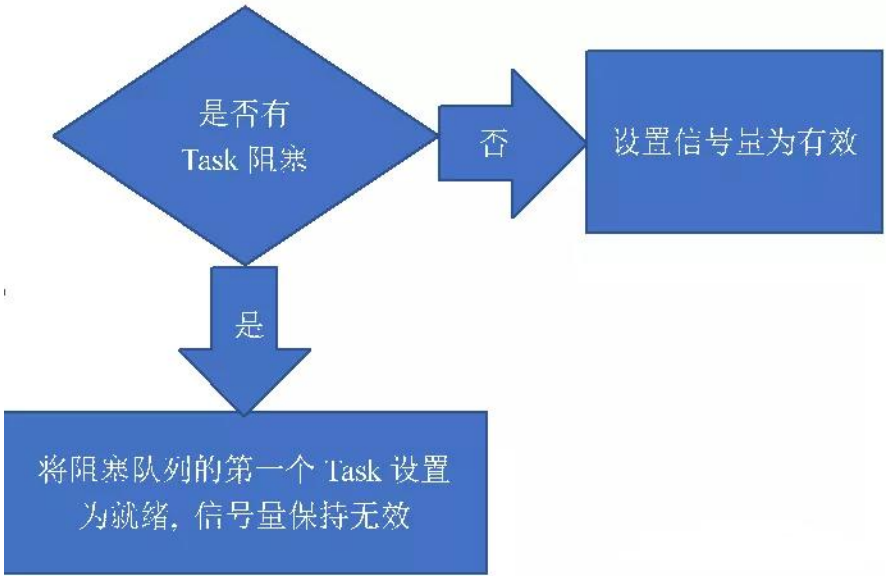
```
-> repeat 2,semGive,semId
Task spawned: id = 0x5993280, name = t3
value = 93926016 = 0x5993280
-> task t1 took semaphore
task t1 took semaphore

-> repeat 300,semGive,semId
Task spawned: id = 0x5993280, name = t4
value = 93926016 = 0x5993280
-> task t1 took semaphore
task t1 took semaphore

->
-> show semId

Semaphore Id         : 0x1638b60
Semaphore Name       : N/A
Semaphore Type       : BINARY
Task Queuing         : FIFO
Pended Tasks         : 1
State                : EMPTY
Options              : 0x0      SEM_Q_FIFO
```

可以看到， repeat 的次数大于 2 之后，任务(t1)都是只能收到两次信号量
我们看看 semGive()的操作流程



从上图可以看到，repeat()第一次释放信号量时，它会将阻塞状态的 t1 置为就绪状态。第二次释放时，没有任务阻塞了，于是将信号量置为有效(0->1)，之后再释放时，都是将信号量置有效(1->1)。直到 repeat()执行完毕，就绪状态的 t1 开始执行后续操作，出现第一次打印。然后又可以成功申请一次信号量(1->0)，就有了第二次打印。这之后，信号量就又是无效的了，t1 再次进入了阻塞状态

这就是二进制信号量的特点，它是用来表示事件是否发生了，而不能表示事件发生的次数

如果需要记录事件发生的次数呢？可以试试提高 t1 的优先级

不过 VxWorks 专门提供了用于计数的信号量：计数信号量

```
/* 创建并初始化计数信号量 */
SEM_ID semCCreate
    (
    /* 排队方式, SEM_Q_FIFO(0x0)或SEM_Q_PRIORITY(0x1) */
    int options,
    /* 初值 */
    int initCount
    );


STATUS semCInit
    (
    SEMAPHORE *pSemaphore,
    int        options,
    int        initCount
    );
```

semCCreate()用来动态创建计数信号量，semCInit()用来初始化静态分配的信号量

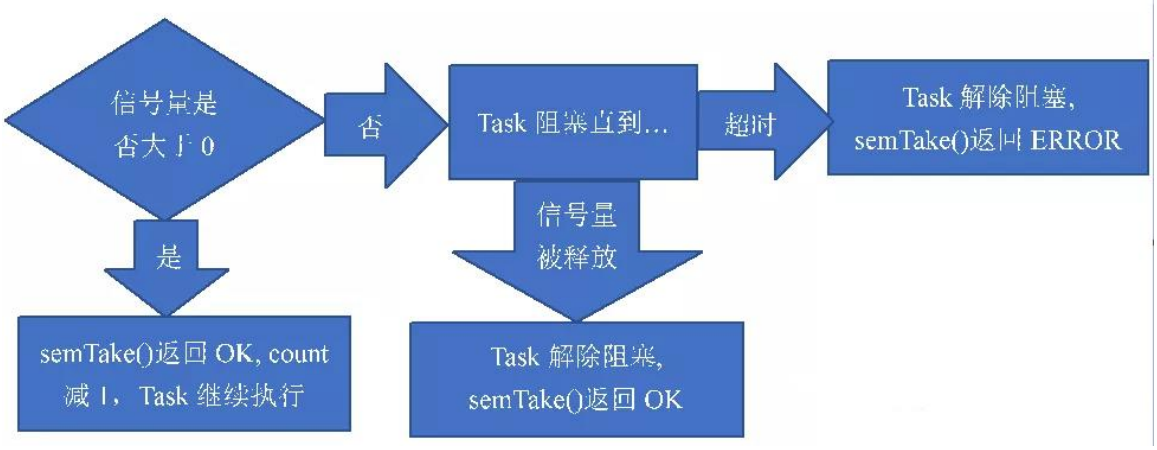
initCount 表示计数信号量的初值，因为是有符号整型值，其取值范围是 0-2147483647(0x0-0x7fffffff)

```
-> semCCreate 0,-1
value = 0 = 0x0
-> printErrno
errno = 0x160069 : S_semLib_INVALID_INITIAL_COUNT.
value = 0 = 0x0
->
-> semCCreate 0,0
value = 23572184 = 0x167aed8
-> semCCreate 0,0x7fffffff
value = 23572312 = 0x167af58 = 'X'
->
-> semCCreate 0,0x80000000
value = 0 = 0x0
-> printErrno
errno = 0x160069 : S_semLib_INVALID_INITIAL_COUNT.
value = 0 = 0x0
->
```

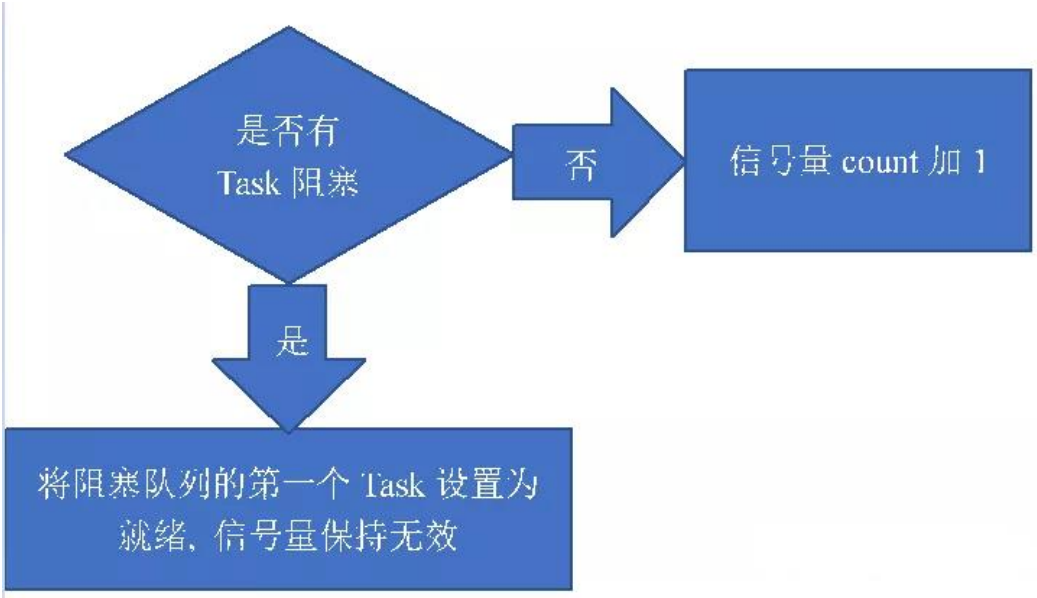
而具体的使用，与二进制信号量非常像

```
STATUS semTake
    (
    SEM_ID semId,
    /* ticks或WAIT_FOREVER(-1U)或NO_WAIT(0) */
    unsigned int timeout
    );
STATUS semGive
    (
    SEM_ID semId
    );
```

semTake()用来申请信号量，信号量无效时，引起阻塞，因此不能在 ISR 中使用



semGive()用来释放信号量，在任务或 ISR 中都可以调用



与二进制信号量不同的是，计数信号量的值不是在 0 和 1 之间变化，而是用一个 count 来记录具体数值。而且目前 count 的值可以超过 2147483647(0x7fffffff)

```
-> semCCreate 0,0x7fffffff
value = 23274408 = 0x16323a8
-> semGive 0x16323a8
value = 0 = 0x0
-> show 0x16323a8

Semaphore Id          : 0x16323a8
Semaphore Name        : N/A
Semaphore Type        : COUNTING
Task Queuing          : FIFO
Pended Tasks          : 0
Count                 : 0
Options               : 0x0      SEM_Q_FIFO
```

超过之后，semGive()和 semTake()还可以正常操作。只是 show()操作时，只能看到低 31 位的值

```
-> semTake 0x16323a8,-1
value = 0 = 0x0
-> semTake 0x16323a8,-1
value = 0 = 0x0
-> show 0x16323a8

Semaphore Id          : 0x16323a8
Semaphore Name        : N/A
Semaphore Type        : COUNTING
Task Queuing          : FIFO
Pended Tasks          : 0
Count                 : 2147483646
Options               : 0x0      SEM_Q_FIFO
```

从源码里可以看到，只有 count 超过 4294967295(0xffffffff)时，才会溢出。看来这应该是 VxWorks 的一个小 bug 了

```
[semCLib.c (src\wind)]
O选项 V视图 W窗口 H帮助
    if (++semId->semCount == 0)      /* give sem, check rollover */
        {
        semId->semCount--;
        OBJ_UNLOCK (semClassId, level);
        errno = S_semLib_COUNT_OVERFLOW;
        return (ERROR);
```

Anyway，实际应用中，count 的值不太可能那么大的。还是回到开始位置，把 testSemB 那个例子改了看看吧

```
#include <semLib.h> /* semCCreate */
#include <stdio.h>  /* printf */
#include <taskLib.h>/* taskName */

SEM_ID semId;

void testSemC()
    {
    semId=semCCreate(SEM_Q_FIFO, SEM_EMPTY);
    while(1)
        {
        semTake(semId,WAIT_FOREVER);

        printf("task %s took semaphore\n", taskName(0));
        }
    }
```

这时候再多次释放信号量，任务(t1)就可以收到多次了

```
-> sp testSemC
Task spawned: id = 0x595e870, name = t1
value = 93710448 = 0x595e870 = 'p'
->
-> repeat 5,semGive,semId
Task spawned: id = 0x38e11f0, name = t2
value = 59642352 = 0x38e11f0
-> task t1 took semaphore
task t1 took semaphore
task t1 took semaphore
task t1 took semaphore
task t1 took semaphore

->
```

同时，计数信号量也支持 semFlush()操作，即它也是可以用于多任务同步的。最后跑一个静态初始化的例子吧

```
#include <semLib.h>  /* semCInit */
#include <stdio.h>   /* printf */
#include <taskLib.h> /* printf */
#include <errnoLib.h>/* errnoGet */

SEMAPHORE mySem;

void testSemC()
    {
    STATUS sta = semCInit(&mySem, SEM_Q_FIFO, SEM_EMPTY);
    if (ERROR == sta)
        {
        printf("semCInit failed\n");
        return;
        }

    sta = semGive(&mySem);
    if (ERROR == sta)
        {
        printf("first semGive failed\n");
        return;
        }

    semTerminate(&mySem);
    sta = semGive(&mySem);
    if (ERROR == sta)
        {
        printf("second semGive failed: 0x%x\n", errnoGet());
        return;
        }
    }
```

mySem 是在编译时就分配空间了，semCInit()里就不用在动态申请内存了

```
-> sp testSemC
Task spawned: id = 0x595e870, name = t1
value = 93710448 = 0x595e870 = 'p'
-> second semGive failed: 0x3d0001

-> printErrno 0x3d0001
errno = 0x3d0001 : S_objLib_OBJ_ID_ERROR.
value = 0 = 0x0
->
```