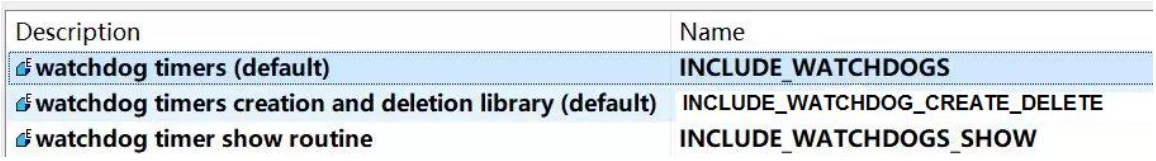
VxWorks 下的看门狗 WatchDog

WatchDog，看门狗，一种定时器。很多 CPU 都带有 WatchDog 功能，这种硬件的 WatchDog 可以在系统死掉时重启系统，让系统回到可以工作的状态。它并不能防止系统死掉，但是能够让系统起死回生，从而提高了系统的可用性。硬件 WatchDog 的局限性是只能在系统范围内生效，不能针对单个任务，某个任务死掉时，WatchDog 没法知道。
VxWorks 系统提供了一种软件的 WatchDog 定时器，工作原理是在系统时钟的 ISR 上关联一个 C 函数。



Description	Name
watchdog timers (default)	INCLUDE_WATCHDOGS
watchdog timers creation and deletion library (default)	INCLUDE_WATCHDOG_CREATE_DELETE
watchdog timer show routine	INCLUDE_WATCHDOGS_SHOW

要想使用这种 WatchDog，需要先创建它。用完之后，可以删除它

```
/*
 * create a watchdog timer
 * RETURNS: The ID for the watchdog created,
 * or NULL if memory is insufficient.
 */
WDOG_ID wdCreate(void);

/*
 * delete a watchdog timer
 * This routine de-allocates a watchdog timer.
 * The watchdog will be removed from the timer queue if it has been started
 * RETURNS: OK, or ERROR if the watchdog timer cannot be de-allocated.
 */
STATUS wdDelete(WDOG_ID wdId);
```

WatchDog 也可以静态实例化

```
/*
 * initialize a watchdog timer
 * This routine initializes a static watchdog or
 * a watchdog embedded in a larger object.
 * RETURNS: OK, or ERROR if the watchdog could not be initialized.
 */
STATUS wdInit(WDOG *pWdog);

/*
 * terminate a watchdog timer
 * This routine terminates a watchdog timer.
 * The watchdog will be removed from the timer queue if it has been started
 * This routine differs from wdDelete() in that
 * associated memory is not de-allocated
 * RETURNS: OK, or ERROR if the watchdog cannot be terminated.
 */
STATUS wdTerminate(WDOG_ID wdId);
```

创建 WatchDog 之后，就可以启动它了

```
/*
 * add a watchdog timer to the system tick queue
 * RETURNS: OK, or ERROR if the watchdog timer cannot be started
 */
STATUS wdStart
    (
    WDOG_ID wdId,
    unsigned int delay,/* 大于0 */
    FUNCPTR pRoutine,
    int parameter
    );
```

wdStart()会将 WatchDog 放入系统 Tick 的工作队列里。当 wdId 的 delay 结束后，系统时钟的 ISR 就会调用 pRoutine(parameter)。参数 delay 的单位是系统时钟的 tick。系统时钟每秒的 tick 数，可以由 sysClkRateGet()来获得。

WatchDog 的大致工作机制如下

```
usrRoot()
    {
    ...
    sysClkConnect((FUNCPTR)usrClock, 0);
    ...
    }

  usrClock()
    -> tickAnnounce()
    -> windTickAnnounce()
    -> WDOG_ID->wdRoutine(WDOG_ID->wdParameter)
```

写一个最简单的例子：

```
#include <stdIo.h>
#include <wdLib.h>
#include <logLib.h>

WDOG_ID myWd=NULL;

void test()
    {
    if(NULL == myWd)
        {
        myWd = wdCreate();
        }
    wdStart(myWd, 60, logMsg, (int)"hello\n");
    }
```

跑一下

```
-> sp test
Task spawned: id = 0x5969870, name = t1
value = 93755504 = 0x5969870 = 'p'
-> interrupt: hello
```

可以看到，WatchDog 的执行是一次性的，即每调用一次 wdStart()，其关联的 C 函数只执行一次。而通常情况下，定时机制都是周期执行的，那么代码可以这样来写

```
#include <stdIo.h>
#include <wdLib.h>
#include <logLib.h>

IMPORT int sysClkRateGet ();

WDOG_ID myWd=NULL;

static void wdISR(int para)
    {
    logMsg("%s\n", para,1,2,3,4,5);
    wdStart(myWd, sysClkRateGet(), (FUNCPTR)wdISR, para);
    }

void test()
    {
    if(NULL == myWd)
        {
        myWd = wdCreate();
        }
    wdStart(myWd, sysClkRateGet(), (FUNCPTR)wdISR,"Hello;\n" );
    }
```

即每次 WatchDog 计时结束，就再次启动同一个 WatchDog。这样就实现了周期调用 logMsg()的功能

```
-> sp test
Task spawned: id = 0x5969870, name = t1
value = 93755504 = 0x5969870 = 'p'
-> interrupt: hello

interrupt: hello

interrupt: hello

interrupt: hello
```

另外，在 WatchDog 计时结束之前，可以随时将其停下

```
/*
 * cancel a currently counting watchdog
 * RETURNS: OK, or ERROR if the watchdog timer cannot be canceled
 */
STATUS wdCancel(WDOG_ID wdId);
```

```
-> sp test
Task spawned: id = 0x5969870, name = t1
value = 93755504 = 0x5969870 = 'p'
-> wdCancel myWd
value = 0 = 0x0
->
```

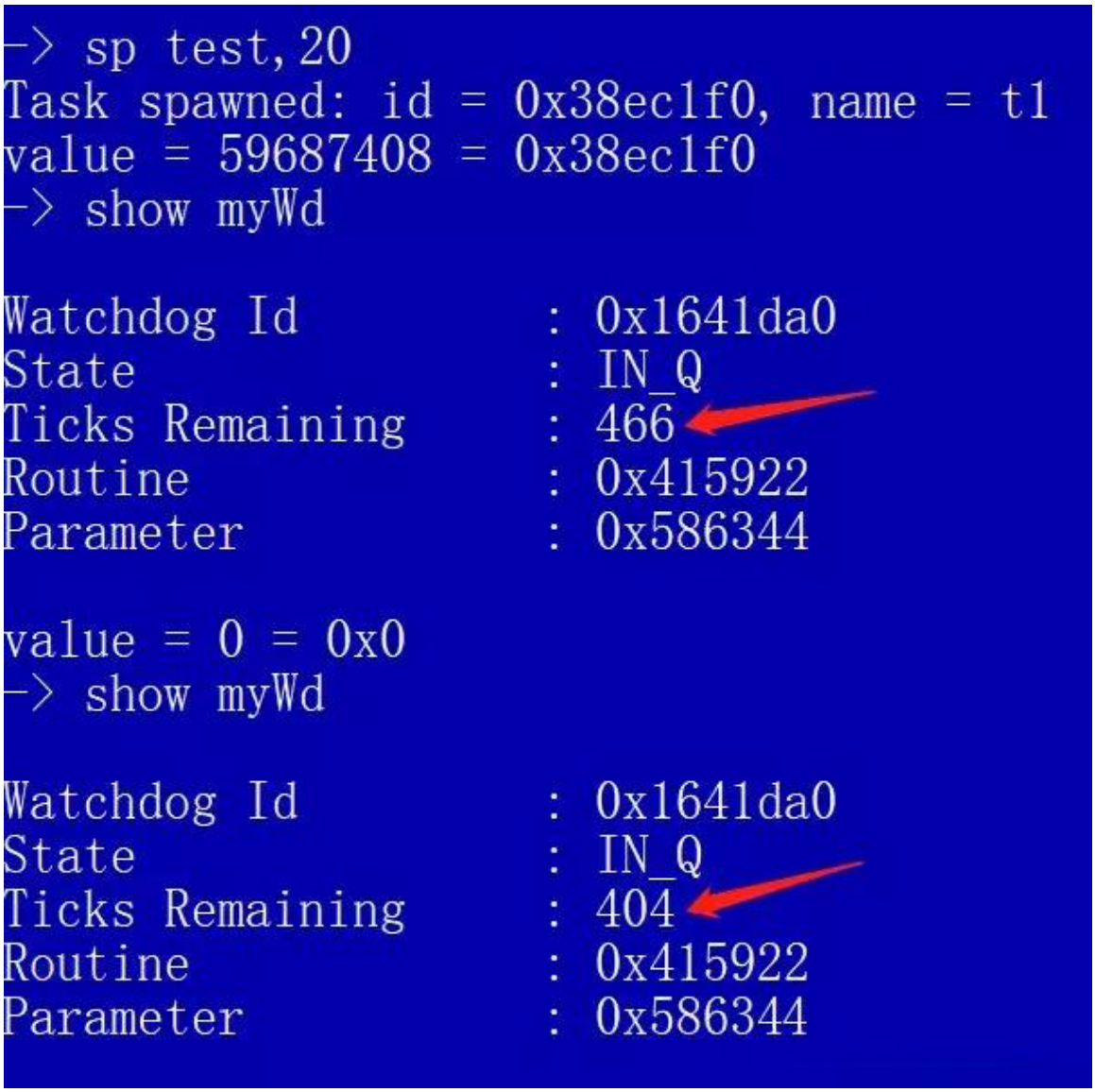
如果使用 wdStart()再次启动同一个 WatchDog，就相遇于将其重置，让其重新开始计时，也就类似于硬件 WatchDog 的作用了

```
#include <stdIo.h>
#include <wdLib.h>
#include <logLib.h>
#include <sysLib.h>
#include <taskLib.h>
#include <rebootLib.h>

WDOG_ID myWd;

static void doMyApp(int num)
    {
    taskDelay(sysClkRateGet()*((num<1)?1:num));
    }

static void wdISR(int para)
    {
    logMsg("%s\n", para,1,2,3,4,5);
    reboot(BOOT_NORMAL);
    }

void test(int num)
    {
    myWd = wdCreate();
    while(1)
        {
        wdStart(myWd, sysClkRateGet()*10,
                (FUNCPTR)wdISR, (int)"doMyApp is timeout\n");
        doMyApp(num);
        }
    }
```

这个例子里，应用程序 doMyApp()的执行时长 num 小于 10 秒时，它就会周期执行。一旦它的执行时长超过 10 秒钟，WatchDog 就会被触

发，在这个例子里，系统就会重启。例如在 Shell 里执行： sp test,20
可以看到，VxWorks 的 WatchDog 比硬件的 WatchDog 更灵活，因为它超时后，不一定就是直接重启，可以自定义超时后的处理机制。

使用 show()命令，可以看到 WatchDog 的状态



当它的 Ticks Remaining 到 0 后，Routine 就被执行了

因为 WatchDog 关联的这个 C 函数是在 ISR 里执行的，所以它不能调用那些可能引起阻塞的机制，例如 printf()，这也是为什么刚刚的例子里都是用的 logMsg()。同样是这个原因，WatchDog 里也不应该执行太多操作，因此用它来定时的话，一般是与任务来配合操作

```
#include <stdIo.h>
#include <wdLib.h>
#include <sysLib.h>
#include <semLib.h>

static WDOG_ID myWd;
static SEM_ID mySem;

static void doMyApp()
    {
    printf("hello\n");
    }

static void wdISR(int para)
    {
    semGive(mySem);
    wdStart(myWd, sysClkRateGet(), (FUNCPTR)wdISR, 0);
    }

void test()
    {
    myWd = wdCreate();
    mySem = semBCreate(SEM_Q_PRIORITY, SEM_EMPTY);
    wdStart(myWd, sysClkRateGet(), (FUNCPTR)wdISR, 0);

    while(1)
        {
        semTake(mySem, WAIT_FOREVER);
        doMyApp();
        }
    }
```

这个例子里，myWd 每秒释放一次信号量，doMyApp()就可以每秒执行一次，而其执行的应用不需要担心 ISR 的限制了，例如可以使用 printf()

```
-> sp test
Task spawned: id = 0x5969870, name = t1
value = 93755504 = 0x5969870 = 'p'
-> hello
hello
hello
hello
hello
```

最后，WatchDog 的数量也不要太多，虽然软件上没有限制，但硬件能力总是有上限的，如果系统时钟里包含了太多操作，对系统性能肯定有影响了

出个小题：
已知 doMyApp(Tb)执行的时长为 Tb，Tb<Ta。问 test1()、test2()、test3()、test4()这 4 个测试用例中，doMyApp()的执行周期分别为多少

```
void test1()
    {
    while(1)
        {
        doMyApp(Tb);
        taskDelay(Ta);
        }
    }
```

```
WDOG_ID myWd;
static void wdISR(int para)
    {
    doMyApp(Tb);
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    }
void test2()
    {
    myWd = wdCreate();
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    }
```

```
WDOG_ID myWd;
static void wdISR(int para)
    {
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    doMyApp(Tb);
    }
void test3()
    {
    myWd = wdCreate();
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    }
```

```
WDOG_ID myWd;
SEM_ID mySem;
static void wdISR(int para)
    {
    semGive(mySem);
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    }
void test4()
    {
    myWd = wdCreate();
    mySem = semBCreate(SEM_Q_FIFO, SEM_EMPTY);
    wdStart(myWd, Ta, (FUNCPTR)wdISR, 0);
    while(1)
        {
        semTake(mySem, WAIT_FOREVER);
        doMyApp(Tb);
        }
    }
```