

## Re: question about thread scheduling

---

*Source:*

<http://www.tech-archive.net/Archive/WindowsCE/microsoft.public.windowsce.platbuilder/2007-03/msg00689.html>

---

- *From:* "Zhiqiang Li" <[zhiqiang.li@xxxxxxxxxxx](mailto:zhiqiang.li@xxxxxxxxxxx)>
  - *Date:* Mon, 19 Mar 2007 10:55:01 +0100
- 

my control thread is triggered only once, not many times(after I have used flag FOREVER to remove the while(!pISTStruct->abort){ } statement). So, I still have problems with rescheduling.

have changed my program to:

```
DWORD WINAPI OptimizationThread( LPVOID lpvParam )
{
// Do all interrupt processing to complete the interaction
// with the board so we can receive another interrupt.
ISTStruct* pISTStruct=(ISTStruct*)lpvParam ;
SYSTEMTIME st;
#ifdef FOREVER
while(!pISTStruct->abort){
#endif
//wait for the 4ms timer interrupt event...
#ifdef SYNCHR_EVENT
WaitForSingleObject(pISTStruct->hEvent, INFINITE);
#endif
if(pISTStruct->abort){
return 0;
}
GetSystemTime(&st);
DEBUGMSG(TRUE, (L"Current SystemTime%d \r\n", st.wMilliseconds));
#ifdef SYNCHR_EVENT
InterruptDone(SYSINTR_SOFT4MS);
#endif
#ifdef FOREVER
}
#endif
return 1;
}
```

I also modify the code inside "ULONG PeRPISR(void) " in fwpc.c to enforce it to return SYSINTR\_RESCHED(here FLAG\_4MS is defined somewhere in oal\_timer.h as" #ifndef FLAG\_4MS #define FLAG\_4MS #endif"):

```
#ifndef FLAG_4MS
//for the scheduler to reschedule every 4ms
ulRet = SYSINTR_RESCHED;
#endif
```

Re: question about thread scheduling

I also have problems with debugging code inside PeRPISR(void), I can not perform actions like step into, and so on.

For example, when I attach the device, the only available item under Debug is Break All, and Stop Debugging. When I click on Break All, it leads me to dbgbrk.c. Then I can click on Start, and nothing happened. I think there must be something wrong with setting breakpoint inside PeRPISR(void), there is only circle with a mark on the line, not a filled point.

thanks in advance

Zhiqiang

"voidcoder" <voidcoder@xxxxxxxx> schrieb im Newsbeitrag  
news:%23IS6h0ZaHHA.1220@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Aren't you, by chance, debug printing every 4ms  
from your thread?

Zhiqiang Li wrote:

thanks for your reply.

Now I am just considering getting one thread running every 4ms, later I will divide my work into more threads, as you have suggested.

I have got a lot of same debug message, it seems that now the thread is rescheduled. But after launching the remote kernel tracker, my debug messages disappear. And I do not know why.

"Henrik Viklund" <henrik.viklund@xxxxxxxx> schrieb im Newsbeitrag  
news:1174209635.140915.224560@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

On 17 Mar, 11:12, "Zhiqiang Li" <zhiqiang...@xxxxxxxx> wrote:

Hi Remi,

thanks for your advice.

Let me tell you what I exactly want to do:

In essential, it is a control problem, and I must send a voltage to the

motor every 4ms(this can be configured to another value in other cases),

and

this voltage is calculated through a process model, which is based on

neural

network. During the 4ms, I must use the CUP resource as much as

possible,

since algorithms on neural networks are always

computational intensive,

and



Re: question about thread scheduling

people to take care of you from the beginning.

If I understood what you want to do, you have implemented a polling loop

to do some job (what kind of job?) every 4ms.

You have put this loop in a high-priority thread started by the

XXX\_Init

function of a stream driver.

You are using CE 6.0 on a CEPC.

The loop should thus look like:

```
while (!g_bSuicide)
{
Sleep(3);
<do some job>
}
```

There is no reason that such a loop exits except when g\_bSuicide is set to

TRUE (something that can be done in XXX\_Deinit) or the <do some job> code

executes a break or goto instruction.

Now, why Sleep(3) instead of Sleep(4)?

Because Sleep(3) will sleep for

\*at

least\* 3 ms and because Sleep is synchronized on timer ticks, that fire every ms (producing the SYSINTR\_RESCHEDED you are focusing on.)

So,

Sleep(3)

returns from sleeping right after the third tick and a new 1ms slice

begins. You <do some job> during a portion of this slice and reenter

Sleep(3). Sleep(3) puts your thread to sleep for 3 timer ticks and that

means the time remaining in the current tick + 3 ms.

HTH

Remi- Dölj citerad text –

– Visa citerad text –

Re: question about thread scheduling