

## Re: Alternative to a thread Thread in MFC??

**Source:** <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2004-07/1844.html>

---

**From:** Joseph M. Newcomer (*newcomer\_at\_flounder.com*)

**Date:** 07/24/04

Date: Sat, 24 Jul 2004 02:58:55 -0400

See below...

Bottom line, you have a completely unsalvageable mess and both the device driver and the application code require a total rewrite. Nothing with an interface this bad could ever be made to work. You are asking how to fine-tune something that is so deeply and fundamentally flawed that nothing short of a complete rewrite could ever produce something usable. You don't "fine-tune" something this bad, you scrap it.

joe

On Fri, 23 Jul 2004 22:44:02 -0700, "Vijay" <vikramuv@blr.philips.com> wrote:

>Is the way of Creating thread Right??

>

>or the main program getch() is adding more delays???

>how to stop a long lasting process,,,

>like mapping Ctrl+c which kills application...can i map few keys and trap it rather than wait in the main program.

>

>

>

>

>main()

>{

> while(1)

> {

>

> c= getch()

> switch(s)

> {

> case 's': // start

> CreateThread(NULL,0,ThreadProc, NULL,0,NULL);

> break;

> }

>

>}

>

>DWORD WINAPI ThreadProc(LPVOID lpVoid)

>{

```
>  
> DWORD dwStatus;  
> RETURN_CODE ReturnCode;  
>  
> repeat = true;  
> while(repeat)  
****
```

Given that you are not resetting repeat inside the loop, why create the variable at all?

```
****  
> { /* loop */  
>  
> dwStatus = WaitForSingleObject( *(pstrHsrsDevice->puFifoEventHandler),  
> INFINITE);  
****
```

Weird naming convention. What is a pointer to string doing pointing to an event? (pstr means a pointer to a string, period. It amazes me that people continue to use these naming conventions in ways that make no sense. Lose the bogus naming convention, since it clearly obscures what is going on. Another triumph of Hungarian Notation over Common Sense).

```
*****  
> switch(dwStatus)  
> { /* dwStatus */  
> case WAIT_ABANDONED:  
> printf("WAIT_ABANDONED\n");  
****
```

This can only happen for a mutex, so why are you using it for an event?

```
****  
> break;  
>  
> case WAIT_OBJECT_0:  
> {  
> // Interrupt recieved...  
****
```

You still talk about "interrupt received", although there is no mechanism in Windows application level to deal with such a concept. What does this mean, really? Who sets the event? Who resets it? Trying to make sense out of code that is talking to an undocumented interface is nearly impossible, but I'm guessing what the interface is likely to be doing. Even my most optimistic guess suggests that the whole interface and its associated device driver should be scrapped.

```
****  
> // Do not accept further interrupts untill data is transferred  
>  
> // Set Usero to zero  
> ReturnCode = HsrsInterruptHandling (false,pstrHsrsDevice);  
> if(ReturnCode != ApiSuccess)  
> return ReturnCode;  
> // Start Data transfer  
> ReturnCode = HsrsISR(pstrHsrsDevice);  
> if(ReturnCode == ApiDmaDone)  
> {  
> printf("\n\nDMA transfer Complete ");
```

```
> return ApiSuccess;
> }
> else
> if(ReturnCode != ApiSuccess)
> {
> DisplayResult(" Error HsrsISR ",ReturnCode);
> return ReturnCode;
> }
> //
> // Data transfer is complete
> // Set Usero to 1
> // Wait for Pci Interrupt
> //
****
```

Waiting for an interrupt is done in a device driver. It makes no sense to even consider this as a concept at user space. I think there is some bizarre confusion here about what constitutes a device driver. If you want to read something, you do a ReadFile, or a DeviceIoControl, and that device driver handles all the issues of enabling/disabling interrupts, tweaking the device registers, handling the buffering, etc. This code suggests some sort of weird ad-hoc mess that probably can't even handle multiple requests or thread safety. If you paid for a product that has this interface, you have been seriously ripped off. If you wrote it, consider scrapping it and doing a proper device driver.

I don't think any of this code is salvageable, because the mess it is interfacing to is unsalvageable. You have a program that doesn't work because it is interfacing to something that cannot possibly make sense, and in ways that cannot possibly be made to work under even the most optimistic scenarios. And certainly will fail in any real environment.

```
****
>
> ReturnCode = HsrsInterruptHandling (true,pstrHsrsDevice);
> if(ReturnCode != ApiSuccess)
> {
> return ReturnCode;
> }
> }
> break;
>
> case WAIT_TIMEOUT:
> printf("WAIT_TIMEOUT\n");
> break;
>
> default:
> printf("\nError ..check event..");
> return 0;
> } /* dwStatus */
> } /* loop */
>
> return 0;
> }
> // ThreadProc
>
```

```
>
>
>RETURN_CODE HsrsISR(uHsrsDevice *pstrDevice )
>{
>
> // call scatter gather DMA
>
> ReturnCode = HSRS_inSglDma(
> pstrDevice->uplxHandle,
> &(pstrDevice->pusData_to_transfer[ulDataIndex]),
> ulTransferCount
> );
>
>
> return ReturnCode;
>
>}
>
> // instead of create Thread .. i had replaced the code in thread Proc without thread
****
```

This code was so badly indented it is almost unreadable. I had to re-indent it just to make sense of it.

\*\*\*\*

```
>
>main()
>{
> while(1)
> {
>
> c= getch();
> switch(s)
> { /* s */
> case 's': // start
> repeat = true;
> while(repeat)
> { /* loop */
>
> dwStatus = WaitForSingleObject(
> *(pstrHsrsDevice->puFifoEventHandler),
> INFINITE);
> switch(dwStatus)
> { /* dwStatus */
> case WAIT_ABANDONED:
> printf("WAIT_ABANDONED\n");
> break;
>
>
****
```

Makes no sense. This only applies to mutex objects

\*\*\*\*

```
> case WAIT_OBJECT_0:
> {
```

```
> // Interrupt recieved...
> // Do not accept further interrupts untill data is transferred
****
```

I have no idea what this could mean. It is the responsibility of the device driver to deal with interrupts, and this is application-level code. In fact, once an interrupt is taken, no interrupt should even be POSSIBLE until another I/O transaction is initiated (there are exceptions to this rule, but they usually result in gratuitously complex drivers). Enabling and disabling interrupts must be done in the device driver, and would never be done from application level. It doesn't make sense, and it cannot possibly work because of scheduling latencies. I have no idea what sort of bizarre device driver you are using, but if you have to enable/disable interrupts from application space, the best thing that could be done with it is to scrap it. I would suggest burying the source code in a toxic waste site.

\*\*\*\*\*

```
>
> // Set Usero to zero
> ReturnCode = HsrsInterruptHandling (false,pstrHsrsDevice);
> if(ReturnCode != ApiSuccess)
> return ReturnCode;
****
```

This scares me. Reading this line I think confirms beyond any shadow of a doubt that the interface to this device is a hopeless mess. Has it occurred to you that the delay between "taking an interrupt" and executing this code could be measured in SECONDS? Not milliseconds, not microseconds, but SECONDS.

\*\*\*\*

```
>
> // Start Data transfer
> ReturnCode = HsrsISR(pstrHsrsDevice);
****
```

This code makes no sense at user space. Issue a ReadFile or DeviceIoControl. This interface suggests a driver whose source code should only be edited while wearing a Level 4 Hazmat suit, in a Class 4 Biohazard room ventilated by a high-grade HEPA filter.

\*\*\*\*

```
> if(ReturnCode == ApiDmaDone)
> {
> printf("\n\nDMA transfer Complete ");
> return ApiSuccess;
> }
> else
> if(ReturnCode != ApiSuccess)
> {
> DisplayResult(" Error HsrsISR ",ReturnCode);
> return ReturnCode;
> }
> //
> // Data transfer is complete
> // Set Usero to 1
> // Wait for Pci Interrupt
> //
> ReturnCode = HsrsInterruptHandling (true,pstrHsrsDevice);
****
```

When you post code like this, it is reasonably important to document what is really going on. The above comments are suggestive of something that is so bizarre that there is not the slightest hope that it could ever be made to work. (I have no idea what a "Usero" is, or what setting it to 1 would do, but the phrase "Wait for Pci Interrupt" scares me silly. Based solely on this one comment I would believe that the driver is an unsalvageable mess, but the rest of the interface seems only to confirm that this is so unbelievably bad that nothing can be done to save it)

\*\*\*\*\*

```
> if(ReturnCode != ApiSuccess)
> {
> DisplayResult("Error HsrsInterruptHandling set to 1 :",ReturnCode);
> return ReturnCode;
> }
```

\*\*\*\*\*

Give that setting or clearing an interrupt from user space makes no sense, the notion that it could FAIL makes even less sense. But even if setting/clearing it could make sense, what could possibly make it NOT succeed?

\*\*\*\*\*

```
> }
> break;
>
> case WAIT_TIMEOUT:
> printf("WAIT_TIMEOUT\n");
> break;
>
> default:
> printf("\nError ..check event..");
> return 0;
> } /* dwStatus */
> } /* loop */
> break;
> } /* s */
>
> }
```

Joseph M. Newcomer [MVP]

email: [newcomer@flounder.com](mailto:newcomer@flounder.com)

Web: <http://www.flounder.com>

MVP Tips: [http://www.flounder.com/mvp\\_tips.htm](http://www.flounder.com/mvp_tips.htm)