

Einfache Threading Klasse

Vor allem "Pendler" zwischen C#, Java und C/C++ dürften öfter Probleme mit der Art und Weise haben, wie Threads unter C/C++ erstellt werden. Da wir immer wieder auf das selbe Problem gestoßen sind, haben wir uns eine einfache Thread Klasse entwickelt, die ganz ähnlich wie unter Hochsprachen wie C# oder Java anzuwenden ist.

Das Herzstück dieser Threading Klasse ist folgendes Template:

```
#ifndef _memberthreads_h
#define _memberthreads_h

template <class T>
struct ThreadData
{
public:
    typedef void (T::*TFunc) ();
    HANDLE hEvent;
    T* pThreadObject;
    TFunc pThreadFunc;
    HANDLE threadHandle;
    static DWORD _ThreadFunc(ThreadData<T>* pThis)
    {
        ThreadData<T> td=*pThis;
        SetEvent(td.hEvent);
        SuspendThread(td.threadHandle);
        ((*td.pThreadObject).*(td.pThreadFunc)) ();
        return 0;
    }
};

template <class T>
HANDLE CreateMemberThread(T* p,void (T::*func) ())
{
    ThreadData<T> td;
    td.pThreadObject=p;
    td.pThreadFunc=func;
    td.hEvent=CreateEvent(NULL,0,0,NULL);

    if(td.hEvent==NULL || GetLastError() == ERROR_ALREADY_EXISTS)
        return NULL;

    DWORD Dummy; //Um win 9x glücklich zu machen mit lpThreadId param
    HANDLE ThreadHandle=CreateThread(NULL,NULL,

(LPTHREAD_START_ROUTINE)ThreadData<T>::_ThreadFunc,
                                &td,CREATE_SUSPENDED,&Dummy);

    td.threadHandle = ThreadHandle;

    ResumeThread(ThreadHandle);
    WaitForSingleObject(td.hEvent,INFINITE);
    ::CloseHandle(td.hEvent);
    return ThreadHandle;
};
```

```
}

#endif
Der folgende Code-Ausschnitt macht nun die eigentliche Arbeit:
#include "MemberThreads.h"

#ifndef __THREADSTATE2__
#define __THREADSTATE2__

#define __MUTEXBEGIN WaitForSingleObject(m_hMutex, INFINITE)
#define __MUTEXEND ReleaseMutex(m_hMutex)

enum ThreadState2{
    _NotInitialised,
    _Initialised,
    _Running,
    _Suspended,
    _Sleeping,
    _Terminated
};

#endif //__THREADSTATE2__

#define _WIN32_WINNT 0x0500

template <class _T_>
class CThread2
{
public:
    CThread2(_T_* p, void(_T_::*pStartFunction)())
    {
        m_ThreadState = _NotInitialised;
        m_Handle = CreateMemberThread<_T_>(p, pStartFunction);
        m_hMutex = CreateMutex(NULL, FALSE, NULL);
        m_hSema = CreateSemaphore(NULL, 0, 2, NULL);

        if(m_Handle != NULL)
        {
            __MUTEXBEGIN;
            m_ThreadState = _Initialised;
            __MUTEXEND;
        }
        else
        {
            __MUTEXBEGIN;
            m_ThreadState = _NotInitialised;
            __MUTEXEND;
        }
    }

    ~CThread2(void){}

    void Suspend()
    {
        //__MUTEXBEGIN;

        if(m_ThreadState != ThreadState2::_NotInitialised && m_ThreadState
        != ThreadState2::_Suspended &&
        m_Handle != NULL)
        {
            m_ThreadState = ThreadState2::_Suspended;
        }
    }
};
```

```
        //__MUTEXEND;
        WaitForSingleObject(m_hSema, INFINITE);
        //DWORD retVal = SuspendThread(m_Handle); //der thread suspended
sich selbst!!
        //__MUTEXBEGIN;
        //m_ThreadState = ThreadState2::_Running;
        //__MUTEXEND;
    }
    else
    {
        //__MUTEXEND;
        //printf("Error suspending Thread, since Threadstate does not
match!\n");
    }

}

int Resume()
{
    //__MUTEXBEGIN;

    if(m_Handle != NULL)
    {
        ReleaseSemaphore(m_hSema, 1, NULL);
        m_ThreadState = ThreadState2::_Running;
        return 0;
    }
    else
    {
        //__MUTEXEND;
        return -2;
    }

    //__MUTEXEND;
    //return 0;
}

int Start()
{
    //__MUTEXBEGIN;
    if(m_ThreadState == _Initialised && m_Handle != NULL)
    {
        //__MUTEXEND;

        DWORD retVal = ResumeThread(m_Handle);

        if(retVal == 0)
        {
            return 0;
        }
        else if(retVal == 1)
        {
            __MUTEXBEGIN;
            m_ThreadState = _Running;
            __MUTEXEND;
            return 0;
        }
        else
        {
            __MUTEXBEGIN;
            m_ThreadState = _Suspended;
            __MUTEXEND;
        }
    }
}
```

```
        return -1;
    }
}
else
{
    /*__MUTEXEND;*/

    if(m_Handle == NULL)
    {
        printf("<Thread2.h>m_Handle==NULL\n");
    }
    else
    {
        printf("<Thread2.h>ThreadState2!=Initialized\n");
    }
    return -2;
}
return 1;
}

int Terminate()
{
    if(m_ThreadState != _NotInitialised && m_ThreadState != _Terminated
&&
        m_Handle != NULL)
    {
        if(TerminateThread(m_Handle,0))
            return 0;
        else
            return -1;
    }
    else
        return -2;

    return 0;
}

ThreadState2 GetThreadState(){__MUTEXBEGIN;ThreadState2 ts=
m_ThreadState;__MUTEXEND;return ts;}
HANDLE m_Handle;
private:
    ThreadState2 m_ThreadState;
    HANDLE m_hMutex;
    HANDLE m_hSema;
};
```

Mit dieser Klasse ist es nun sehr einfach möglich einen Thread zu starten:

```
void ZielKlasse::MethodeXY()
{
    CThread2<ZielKlasse>* neuerThread = new
CThread2<ZielKlasse>(this, &CThread2::StartMethode);
    neuerThread->Start();
}

void ZielKlasse:StartMethode()
{
    //Irgendwas tun...
}
```

Bei Fragen zu diesem Sourcecode benutzen Sie bitte das [Forum](#).