Archived:RThread - Symbian threading example

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The content is still valid (to at least Nokia Belle), but is better covered by the Fibonacci1-3: asynchronous programming with active objects examples in the SDK.

Overview

This RThread example shows how to:

- Create a simple CMyThread thread class that runs RThread
- How to listen for thread execution
- How add cleanup stack support for threads
- How add active scheduler support for threads

The CMyThread thread runs the CPeriodic timer.

MMP file

RThread needs

```
LIBRARY euser.lib
```

Header

```
#include <e32base.h>

class MThreadExecuteObserver
    {
    public:
        virtual void ThreadExecuted(TInt aError) = 0;
    };

class CMyThread : public CActive, public MThreadExecuteObserver
    {
    public:
        static CMyThread* NewL(MThreadExecuteObserver& aObserver);
        virtual ~CMyThread();
        TInt ExecuteThread(TTimeIntervalMicroSeconds32 anInterval);
        inline TTimeIntervalMicroSeconds32 Interval(){return iInterval;};

    private: // From MThreadExecuteObserver
        void ThreadExecuted(TInt aError);

    private: // From CActive
        void RunL();
        void DoCancel();

    private:
        CMyThread(MThreadExecuteObserver& aObserver);
        void ConstructL();
        void CreateThreadL();
        TInt StartThread();
        static TInt ThreadFunction(TAny* aParams);
        static TInt PeriodicTick(TAny* aObject);

    private:
        MThreadExecuteObserver&         iObserver;
        RThread iThread;
        TTimeIntervalMicroSeconds32 iInterval;
    };
```

Source

```
#include "CMyThread.h"

const TInt KStackSize = 16384;
_LIT(KExecThreadBaseName, "CMyThread");

// Global thread id counter for making CMyThread thread names unique.
// This is achieved by appending counter to the end of thread name and
// incrementing counter value for next thread.
// This is writable static data.
TInt g_thread_id_counter = 0;

LOCAL_C void GenerateUniqueName(TDes& aResult, CMyThread* aExecutor)
    {
    _LIT(KHexFormat, "_0x%x");
    _LIT(KCounterFormat, "_%d");
    aResult.Copy(KExecThreadBaseName);
    aResult.AppendFormat(KHexFormat, aExecutor);
    g_thread_id_counter++;
    aResult.AppendFormat(KCounterFormat, g_thread_id_counter);
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }

CMyThread* CMyThread::NewL(MThreadExecuteObserver& aObserver)
    {
    CMyThread* self = new (ELeave) CMyThread(aObserver);
    CleanupStack::PushL(self);
    self->ConstructL();
    CleanupStack::Pop(self);
    return self;
    }

CMyThread::CMyThread(MThreadExecuteObserver& aObserver)
    : CActive(EPriorityStandard),iObserver(aObserver)
    {
    CActiveScheduler::Add(this);
    }

CMyThread::~CMyThread()
    {
    Cancel();
    iThread.Close();
    }

void CMyThread::ConstructL()
    {
    }

TInt CMyThread::ExecuteThread(TTimeIntervalMicroSeconds32 anInterval)
    {
    TInt ret = KErrNone;
    iInterval = anInterval;
    TRAP(ret,CreateThreadL());
    if (!ret)
        {
        ret = StartThread();
        }
    return ret;
    }

TInt CMyThread::StartThread()
    {
    TInt ret = KErrNone;
    if(!IsActive())
        {
        // Requests notification when this thread dies
        // normally or otherwise
        iThread.Logon(iStatus);
        SetActive();
        iThread.Resume();
        }
    else
        {
        ret = KErrOverflow;
        }
    return ret;
    }

void CMyThread::ThreadExecuted(TInt aError)
    {
    iObserver.ThreadExecuted(aError);
    }

void CMyThread::RunL()
    {
    iObserver.ThreadExecuted(iStatus.Int());
    }

void CMyThread::DoCancel()
    {
    iThread.LogonCancel(iStatus);
    iThread.Kill(KErrCancel);
    }

void CMyThread::CreateThreadL()
    {
    HBufC* threadName = HBufC::NewLC(KMaxFileName);
    TPtr ptr = threadName->Des();
    GenerateUniqueName(ptr, this);

    User::LeaveIfError(iThread.Create(
                        *threadName,
                        CMyThread::ThreadFunction,
                        KStackSize,
                        NULL,
                        this));

    CleanupStack::PopAndDestroy(threadName);
    }

TInt CMyThread::ThreadFunction(TAny* aParams)
    {
    // 1. Add cleanup stack support.
    CTrapCleanup* cleanupStack = CTrapCleanup::New();

    // 2. Get pointer to thread host
    CMyThread* host = (CMyThread*)aParams;

    TRAPD(err,
        // 3. Add support for active objects
        CActiveScheduler* activeScheduler = new (ELeave) CActiveScheduler;
        CleanupStack::PushL(activeScheduler);
        CActiveScheduler::Install(activeScheduler);

        // 4. Create and start CPeriodic class that is executed in this thread
        CPeriodic* periodic = CPeriodic::NewL(CActive::EPriorityLow);
        CleanupStack::PushL(periodic);
        periodic->Start(host->Interval(),host->Interval(),
        TCallBack(host->PeriodicTick, host));

        // NOTE: When adding CActiveScheduler support for threads we have to
        // add atleast one active object in it or it fails on
        // CActiveScheduler::Start().
        // CPeriodic is derived from CActive active object so that is good for
        // this example.

        // 5. --> Thread execution starts
        CActiveScheduler::Start();
        // 6. --> Thread execution ends (waiting for CActiveScheduler::Stop())

        CleanupStack::PopAndDestroy(periodic);
```


(C) Copyright Nokia 2013. All rights reserved.

```
        CleanupStack::PopAndDestroy(activeScheduler);
        );

    host->ThreadExecuted(err);
    delete cleanupStack;
    return KErrNone;
    }

TInt CMyThread::PeriodicTick(TAny* aObject)
    {
    CMyThread* mythread = (CMyThread*)aObject;
    if (mythread)
        {
        // Thread is executed once so time to stop it
        CActiveScheduler::Stop();
        // After this execution continues from CActiveScheduler::Start()
        }
    // Does not continue again
    // Note: Does not work with this CPeriodic class
    return EFalse;
    }
```

Running the CMyThread

Place `CMyThread` into class member variable and execute it

```
TTimeIntervalMicroSeconds32 time(100);
iMyThread = CMyThread::NewL(*this);
iMyThread->ExecuteThread(time);
```

When the thread is executed and it calls its observer that is a reference parameter in `CMyThread::NewL(*this)`

```
void CYourClass::ThreadExecuted(TInt aError)
    {
    // Will be called after thread execution
    }
```

Postconditions

The thread has been created and executed.

COPY-PASTE the desired platforms into the wiki text and save. All relevant version categories should be applied.
Windows Phone: [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

Nokia Asha: [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

Series 40: [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]
[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

Symbian: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]]
[[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.