How to create a server from scratch

This code shows how to create a server from scratch. It shows how to implement the server class, that represents the server itself, and the session, that is created for each communication channel opened from client to server.

29 Nov 2009

Overview

This example shows how to implement a server from scratch step-by-step. It assumed previous knowledge related to the client/server framework (ex. Client-Server Framework).

This example deals with the synchronous requests only. Asynchronous request can be found in Implementing asynchronous requests.

The client counterpart implementation of the server can be viewed here Client side implementation of a server.

Types

1.System Server: Starts with the mobile phone, essential for the OS (e.g. File Server, Window Server). Unexpected termination usually forces phone reboot.

2.Application / Transient Server: Started on request of an application, closed when last client disconnects.If a second client starts a new server, no error is produced, existing server instance is used.

3.Other Servers: Started and closed with an individual application. If used by multiple applications: each app. has its own private instance; e.g. POSIX server.

Server Implementation

A server implementation is based mainly on two objects:

1. The server object, of `CServer2` type.
2. The sessions objects, of `CSession2` type.

Servers usually run in their own process, but at least in a separate thread. Because of this, a server has to install the clean-up stack and the Active Scheduler for the thread it is going to run in. So the first thing to do is to install the clean-up stack and then the active scheduler. The following code shows how to do this.

First step: Install the cleanup stack

```
CTrapCleanup* cleanupStack = CTrapCleanup::New();
if ( !( cleanupStack ) )
   {
    // cannot go on. Need to panic the server. We cannot live without cleanup stack :)
   }
// initialize and starts active scheduler
```

Second step: Install and start active scheduler

```
// Construct active scheduler
CActiveScheduler* activeScheduler = new ( ELeave ) CActiveScheduler;
//We can use the stack here. It is alrady installed!
CleanupStack::PushL( activeScheduler );

// Install active scheduler
CActiveScheduler::Install( activeScheduler );

//Instantiate server object here!

//Starts the scheduler. Thread remains in this loop until the process is terminated.
CActiveScheduler::Start();
```

Third step: Specify and create the server object Before starting the scheduler it is necessary to create the server object. The class that specifies the server object is derived from the `CServer2` class (from Symbian OS v9.x onwards). The main goals of the server are to keep track of the number of sessions that are opened to it and to create new session objects for each opened session with the clients. Due to this, the main functions of the server class are `IncrementSessions`, `DecrementSessions` and `NewSessionL`. The following code shows an implementation of the server class. It was obtained from `S60Ex` folder of SDK named `ClientServerSync`.

The header file.

```
/**
* CTimeServer.
*  An instance of class CTimeServer is the main server class
*  for the ClientServerSync example application
*  Just the main functions are showed here!
*/
class CTimeServer : public CServer2
    {
    public: // New functions
```


(C) Copyright Nokia 2013. All rights reserved.

```
        /**
        * ThreadFunction.
        * Main function for the server thread.
        * @param aNone Not used.
        * @return Error code.
        */
        static TInt ThreadFunction( TAny* aNone );

        /**
        * IncrementSessions.
        * Increments the count of the active sessions for this server.
        */
        void IncrementSessions();

        /**
        * DecrementSessions.
        * Decrements the count of the active sessions for this server.
        * If no more sessions are in use the server terminates.
        */
        void DecrementSessions();

    private: // New methods

        /**
        * ThreadFunctionL.
        * Second stage startup for the server thread.
        */
        static void ThreadFunctionL();

    private: // Functions from base classes

        /**
        * From CServer, NewSessionL.
        * Create a time server session.
        * @param aVersion The client version.
        * @param aMessage Message from client.
        * @return Pointer to new session.
        */
        CSession2* NewSessionL( const TVersion& aVersion,
                                const RMessage2& aMessage ) const;

    };
```

The source code.

```
// ---------------------------------------------------------------------------
// CTimeServer::NewSessionL()
// Creates a time server session.
// Just the main functions are showed here!
// ---------------------------------------------------------------------------
//
CSession2* CTimeServer::NewSessionL( const TVersion& aVersion,
                                     const RMessage2& /*aMessage*/ ) const
    {
    // Check we are the right version
    if ( !User::QueryVersionSupported( TVersion( KTimeServMajorVersionNumber,
                                                 KTimeServMinorVersionNumber,
                                                 KTimeServBuildVersionNumber ),
                                       aVersion ) )
        {
        User::Leave( KErrNotSupported );
        }

    // Make new session
    return CTimeServerSession::NewL( *const_cast<CTimeServer*> ( this ) );
    }

// ---------------------------------------------------------------------------
// CTimeServer::IncrementSessions()
// Increments the count of the active sessions for this server.
// ---------------------------------------------------------------------------
//
void CTimeServer::IncrementSessions()
    {
    iSessionCount++;
    }

// ---------------------------------------------------------------------------
// CTimeServer::DecrementSessions()
// Decrements the count of the active sessions for this server.
// ---------------------------------------------------------------------------
//
void CTimeServer::DecrementSessions()
    {
    iSessionCount--;
    if ( iSessionCount <= 0 )
        {
        CActiveScheduler::Stop();
        }
    }

// ---------------------------------------------------------------------------
// CTimeServer::ThreadFunctionL()
// Second stage startup for the server thread.
// Creates and starts the active scheduler
// ---------------------------------------------------------------------------
//
void CTimeServer::ThreadFunctionL()
    {
    // Construct active scheduler
    CActiveScheduler* activeScheduler = new ( ELeave ) CActiveScheduler;
    CleanupStack::PushL( activeScheduler );

    // Install active scheduler
    // We don't need to check whether an active scheduler is already installed
    // as this is a new thread, so there won't be one
    CActiveScheduler::Install( activeScheduler );

    // Construct our server, leave it on clean-up stack
    CTimeServer::NewLC();    // Anonymous

    RSemaphore semaphore;
    User::LeaveIfError( semaphore.OpenGlobal( KTimeServerSemaphoreName ) );

    // Semaphore opened ok
    semaphore.Signal();
    semaphore.Close();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    // Start handling requests
    CActiveScheduler::Start();

    CleanupStack::PopAndDestroy( 2, activeScheduler ); //Anonymous CTimeServer
    }

// ---------------------------------------------------------------------------
// CTimeServer::ThreadFunction()
// Main function for the server thread.
// Creates the cleanup stack
// ---------------------------------------------------------------------------
//
TInt CTimeServer::ThreadFunction( TAny* /*aNone*/ )
    {
    CTrapCleanup* cleanupStack = CTrapCleanup::New();
    if ( !( cleanupStack ) )
        {
        PanicServer( ECreateTrapCleanup );
        }

    TRAPD( err, ThreadFunctionL() );
    if ( err != KErrNone )
        {
        PanicServer( ESrvCreateServer );
        }

    delete cleanupStack;
    cleanupStack = NULL;

    return KErrNone;
    }

// ---------------------------------------------------------------------------
// E32Main()
// Provides the API for the operating system to start the executable.
// Returns the address of the function to be called.
// ---------------------------------------------------------------------------
//
TInt E32Main()
    {
    return CTimeServer::ThreadFunction( NULL );
    }
```

Session implementation

Fourth step: Specifies the session class Every time a client connects to a server a session object is created in each side. In the server side, this objects manages the connection among server and client. It is responsible for receiving and answering client's requests. This object is derived from the class CSession2 and needs to implement the method ServiceL from which the client's requests arrives. This is the main function of this object. The following code shows an example of session class implementation. Observe that the creation and deletion of each session object implies in a call to IncrementSessions and DecrementSessions of the server object.

The header file.

```
/**
* CTimeServerSession.
*  An instance of class CTimeServerSession is created for each client.
*  Just the main functions are showed here!
*/
class CTimeServerSession : public CSession2
    {
    public: // Constructors and destructors
        /**
        * NewL.
        * Two-phased constructor.
        * @param aServer The server.
        * @return Pointer to created CTimeServerSession object.
        */
        static CTimeServerSession* NewL( CTimeServer& aServer );

        /**
        * NewLC.
        * Two-phased constructor.
        * @param aServer The server.
        * @return Pointer to created CTimeServerSession object.
        */
        static CTimeServerSession* NewLC( CTimeServer& aServer );

        /**
        * ~CTimeServerSession.
        * Destructor.
        */
        virtual ~CTimeServerSession();

    public: // Functions from base classes

        /**
        * From CSession, ServiceL.
        * Service request from client.
        * @param aMessage Message from client
        *                 (containing requested operation and any data).
        */
        void ServiceL( const RMessage2& aMessage );
    };
```

The source code.

```
// ---------------------------------------------------------------------------
// CTimeServerSession::ConstructL()
// Symbian 2nd phase constructor can leave.
// Increments the server sessions.
// Just the main functions are showed here!
// ---------------------------------------------------------------------------
//
void CTimeServerSession::ConstructL()
    {
    iServer.IncrementSessions();
    }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }

// ---------------------------------------------------------------------------
// CTimeServerSession::~CTimeServerSession()
// Destructor.
// Decrement the number of server sessions.
// ---------------------------------------------------------------------------
//
CTimeServerSession::~CTimeServerSession()
    {
    iServer.DecrementSessions();
    }

// ---------------------------------------------------------------------------
// CTimeServerSession::ServiceL()
// Service request from client.
// ---------------------------------------------------------------------------
//
void CTimeServerSession::ServiceL( const RMessage2& aMessage )
    {
    switch ( aMessage.Function() )
        {
        case ETimeServRequestTime :
            RequestTimeL( aMessage );
            break;

        default:
            PanicClient( aMessage, EBadRequest );
            break;
        }
    aMessage.Complete( KErrNone );
    }

// ---------------------------------------------------------------------------
// CTimeServerSession::RequestTimeL()
// Called as a result of the client requesting the time.
// ---------------------------------------------------------------------------
//

void CTimeServerSession::RequestTimeL( const RMessage2& aMessage )
    {
    TTime time;
    time.HomeTime();

    TPtr8 ptr( reinterpret_cast<TUint8*>( &time ), sizeof( time ),
               sizeof( time ) );

    // Write time data to the descriptor which is the first message argument
    aMessage.WriteL( 0, ptr, 0 );
    }
```

MMP file (optional)

The following capabilities are required:

CAPABILITY none

Conclusions

This code shows step-by-step the implementation of a server. This example shows the server-side implementation. This implementation would generate an EXE binary (the server side). For the clients to use this server a client side implementation is also necessary. Usually this implementation come as a DLL. An example of how to implement this DLL can be found here (link to client side implementation).

Related links

- Client side implementation of a server
- Client-Server Framework
- Inter Process Communication in Symbian

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.