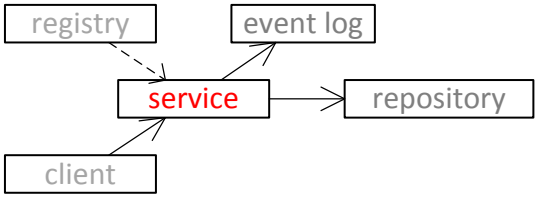
PB173 - Systémové programování Windows

Úkol 10



Rozšiřte projekt Service tak:

- Aby se pomoci parametru "/i" zaregistroval jako služba a
- aby byl služba. Službu pojmenujte „Service.JmenoPrijmeni".

Nutné úpravy (komunikace v rámci HKCU):

- Jména synchronizačních objektů a i namapovaná paměť musí začínat "Global\".
- Pro ověření komunikace s klientem je potřeba službu nastavit tak, aby běžela pod účtem uživatele pod kterým běží klient. Provedete to takto:
    1. v services.msc klikněte pravým tlačítkem myši na vaši službu.
    2. Zvolte volbu „Properties".
    3. Přejdete na záložku „Log On".
    4. Zatrhněte „this account" a vyplňte přihlašovací údaje.
- Pozn. Klienta je nutné pouštět s administrátorským oprávněním.
- Pozn. Komunikace přes data_seg od této chvíle již nebude funkční

Tipy:

- Služba zdroje vytváří klient otevírá

Příloha 1 - Svc.cpp

Zdroj: http://msdn.microsoft.com/en-us/library/windows/desktop/bb540475(v=vs.85).aspx

The following is a complete service sample. When using this code as a template, simply add code to the sections that are prefaced by `TO_DO`.
When building the sample, be sure to link with Kernel32.lib and Advapi32.lib. The file Sample.h is generated when building the resource-only DLL, Sample.dll. For more information, see Sample.mc.

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "sample.h"

#pragma comment(lib, "advapi32.lib")

#define SVCNAME TEXT("SvcName")

SERVICE_STATUS          gSvcStatus;
SERVICE_STATUS_HANDLE   gSvcStatusHandle;
HANDLE                  ghSvcStopEvent = NULL;

VOID SvcInstall(void);
VOID WINAPI SvcCtrlHandler( DWORD );
VOID WINAPI SvcMain( DWORD, LPTSTR * );

VOID ReportSvcStatus( DWORD, DWORD, DWORD );
VOID SvcInit( DWORD, LPTSTR * );
VOID SvcReportEvent( LPTSTR );

//
// Purpose:
//   Entry point for the process
//
// Parameters:
//   None
//
// Return value:
//   None
//
void __cdecl _tmain(int argc, TCHAR *argv[])
{
    // If command-line parameter is "install", install the service.
    // Otherwise, the service is probably being started by the SCM.
    if( lstrcmpi( argv[1], TEXT("install")) == 0 )
    {
        SvcInstall();
        return;
    }

    // TO_DO: Add any additional services for the process to this table.
    SERVICE_TABLE_ENTRY DispatchTable[] =
    {
        { SVCNAME, (LPSERVICE_MAIN_FUNCTION) SvcMain },
        { NULL, NULL }
    };

    // This call returns when the service has stopped.
    // The process should simply terminate when the call returns.
    if (!StartServiceCtrlDispatcher( DispatchTable ))
    {
        SvcReportEvent(TEXT("StartServiceCtrlDispatcher"));
    }
}
```

```
//
// Purpose:
//   Installs a service in the SCM database
//
// Parameters:
//   None
//
// Return value:
//   None
//
VOID SvcInstall()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;
    TCHAR szPath[MAX_PATH];

    if( !GetModuleFileName( "", szPath, MAX_PATH ) )
    {
        printf("Cannot install service (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the SCM database.
    schSCManager = OpenSCManager(
        NULL,                    // local computer
        NULL,                    // ServicesActive database
        SC_MANAGER_ALL_ACCESS);  // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Create the service
    schService = CreateService(
        schSCManager,              // SCM database
        SVCNAME,                   // name of service
        SVCNAME,                   // service name to display
        SERVICE_ALL_ACCESS,        // desired access
        SERVICE_WIN32_OWN_PROCESS, // service type
        SERVICE_DEMAND_START,      // start type
        SERVICE_ERROR_NORMAL,      // error control type
        szPath,                    // path to service's binary
        NULL,                      // no load ordering group
        NULL,                      // no tag identifier
        NULL,                      // no dependencies
        NULL,                      // LocalSystem account
        NULL);                     // no password

    if (schService == NULL)
    {
        printf("CreateService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }
    else printf("Service installed successfully\n");

    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}
```

```
//
// Purpose:
//   Entry point for the service
//
// Parameters:
//   dwArgc - Number of arguments in the lpszArgv array
//   lpszArgv - Array of strings. The first string is the name of
//     the service and subsequent strings are passed by the process
//     that called the StartService function to start the service.
//
// Return value:
//   None.
//
VOID WINAPI SvcMain( DWORD dwArgc, LPTSTR *lpszArgv )
{
    // Register the handler function for the service
    gSvcStatusHandle = RegisterServiceCtrlHandler(
        SVCNAME,
        SvcCtrlHandler);

    if( !gSvcStatusHandle )
    {
        SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));
        return;
    }

    // These SERVICE_STATUS members remain as set here
    gSvcStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    gSvcStatus.dwServiceSpecificExitCode = 0;

    // Report initial status to the SCM
    ReportSvcStatus( SERVICE_START_PENDING, NO_ERROR, 3000 );

    // Perform service-specific initialization and work.
    SvcInit( dwArgc, lpszArgv );
}
```

```
//
// Purpose:
//   The service code
//
// Parameters:
//   dwArgc - Number of arguments in the lpszArgv array
//   lpszArgv - Array of strings. The first string is the name of
//     the service and subsequent strings are passed by the process
//     that called the StartService function to start the service.
//
// Return value:
//   None
//
VOID SvcInit( DWORD dwArgc, LPTSTR *lpszArgv)
{
    // TO_DO: Declare and set any required variables.
    //   Be sure to periodically call ReportSvcStatus() with
    //   SERVICE_START_PENDING. If initialization fails, call
    //   ReportSvcStatus with SERVICE_STOPPED.

    // Create an event. The control handler function, SvcCtrlHandler,
    // signals this event when it receives the stop control code.
    ghSvcStopEvent = CreateEvent(
                         NULL,    // default security attributes
                         TRUE,    // manual reset event
                         FALSE,   // not signaled
                         NULL);   // no name

    if ( ghSvcStopEvent == NULL)
    {
        ReportSvcStatus( SERVICE_STOPPED, NO_ERROR, 0 );
        return;
    }

    // Report running status when initialization is complete.
    ReportSvcStatus( SERVICE_RUNNING, NO_ERROR, 0 );

    // TO_DO: Perform work until service stops.
    while(1)
    {
        // Check whether to stop the service.
        WaitForSingleObject(ghSvcStopEvent, INFINITE);

        ReportSvcStatus( SERVICE_STOPPED, NO_ERROR, 0 );
        return;
    }
}
```

```
//
// Purpose:
//   Sets the current service status and reports it to the SCM.
//
// Parameters:
//   dwCurrentState - The current state (see SERVICE_STATUS)
//   dwWin32ExitCode - The system error code
//   dwWaitHint - Estimated time for pending operation,
//     in milliseconds
//
// Return value:
//   None
//
VOID ReportSvcStatus( DWORD dwCurrentState,
                      DWORD dwWin32ExitCode,
                      DWORD dwWaitHint)
{
    static DWORD dwCheckPoint = 1;

    // Fill in the SERVICE_STATUS structure.
    gSvcStatus.dwCurrentState = dwCurrentState;
    gSvcStatus.dwWin32ExitCode = dwWin32ExitCode;
    gSvcStatus.dwWaitHint = dwWaitHint;

    if (dwCurrentState == SERVICE_START_PENDING)
        gSvcStatus.dwControlsAccepted = 0;
    else gSvcStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;

    if ( (dwCurrentState == SERVICE_RUNNING) ||
           (dwCurrentState == SERVICE_STOPPED) )
        gSvcStatus.dwCheckPoint = 0;
    else gSvcStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the SCM.
    SetServiceStatus( gSvcStatusHandle, &gSvcStatus );
}

//
// Purpose:
//   Called by SCM whenever a control code is sent to the service
//   using the ControlService function.
//
// Parameters:
//   dwCtrl - control code
//
// Return value:
//   None
//
VOID WINAPI SvcCtrlHandler( DWORD dwCtrl )
{
   // Handle the requested control code.
   switch(dwCtrl)
   {
      case SERVICE_CONTROL_STOP:
         ReportSvcStatus(SERVICE_STOP_PENDING, NO_ERROR, 0);

         // Signal the service to stop.
         SetEvent(ghSvcStopEvent);

         return;

      case SERVICE_CONTROL_INTERROGATE:
         break;

      default:
         break;
   }
}
```

```
//
// Purpose:
//   Logs messages to the event log
//
// Parameters:
//   szFunction - name of function that failed
//
// Return value:
//   None
//
// Remarks:
//   The service must have an entry in the Application event log.
//
VOID SvcReportEvent(LPTSTR szFunction)
{
    HANDLE hEventSource;
    LPCTSTR lpszStrings[2];
    TCHAR Buffer[80];

    hEventSource = RegisterEventSource(NULL, SVCNAME);

    if( NULL != hEventSource )
    {
        StringCchPrintf(Buffer, 80, TEXT("%s failed with %d"), szFunction,
GetLastError());

        lpszStrings[0] = SVCNAME;
        lpszStrings[1] = Buffer;

        ReportEvent(hEventSource,        // event log handle
                    EVENTLOG_ERROR_TYPE, // event type
                    0,                   // event category
                    SVC_ERROR,           // event identifier
                    NULL,                // no security identifier
                    2,                   // size of lpszStrings array
                    0,                   // no binary data
                    lpszStrings,         // array of strings
                    NULL);               // no binary data

        DeregisterEventSource(hEventSource);
    }
}
```