Windows 스레드 프로그래밍 [2016.07.07]

작업자 스레드 (Worker Thread)

어떤 단일 작업을 수행하기 위해서 사용하는 것으로 간단히 함수 하나로 이루어진다. 이 함수는 static 클래스 멤버 함수이거나 전역 함수여야 한다. 함수가 종료되면 스레드가 끝난다.

(클래스 객체가 초기화되기 전에 Thread 함수가 호출될 수 있으므로 static 멤버 함수이거나 객체 초기화할 필요가 없는 클래스 밖의 전역 함수여야 함)

```
/**
* ThreadFunc이라는 함수가 스레드로서 주 프로세스와 함께 동시에 실행된다.
* ThreadFunc 함수에서는 보통 for문이나 while문 등의 루프를 돌면서
* 시간이 오래 걸리는 작업을 수행한다.
*/
UINT ThreadFunc(LPVOID pParam) {
 UINT nIteration = (UINT) pParam;
 for(UNIT i=0; i<nIteration; i++)
 {
  // 수행할 작업
 }
}
```

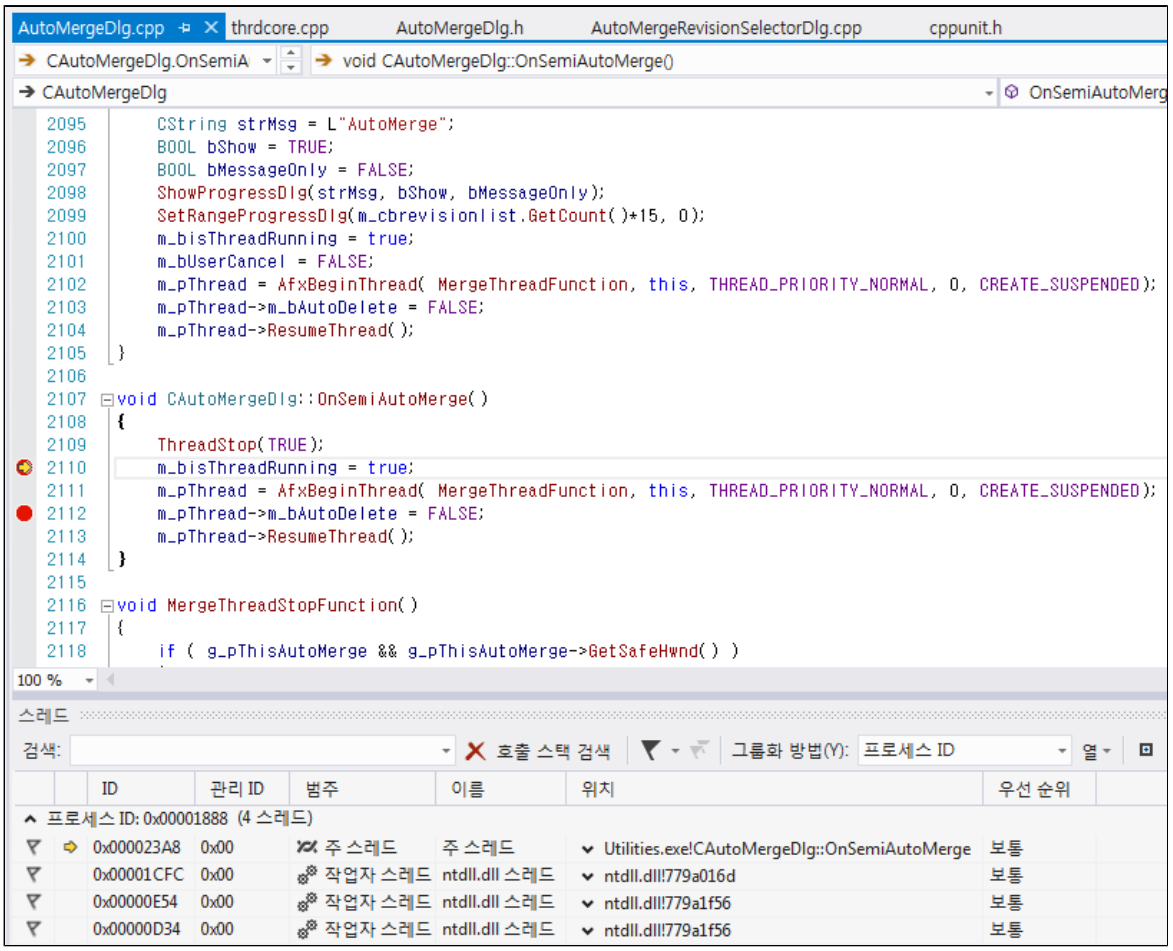
- 예시 - AutoMerge Tool

```
void CAutoMergeDlg::OnSemiAutoMerge() {
 ThreadStop(TRUE);
 m_bisThreadRunning = true;
 m_pThread = AfxBeginThread( MergeThreadFunction, this, THREAD_PRIORITY_NORMAL, 0,
CREATE_SUSPENDED);
 m_pThread->m_bAutoDelete = FALSE;
 m_pThread->ResumeThread();
}
```

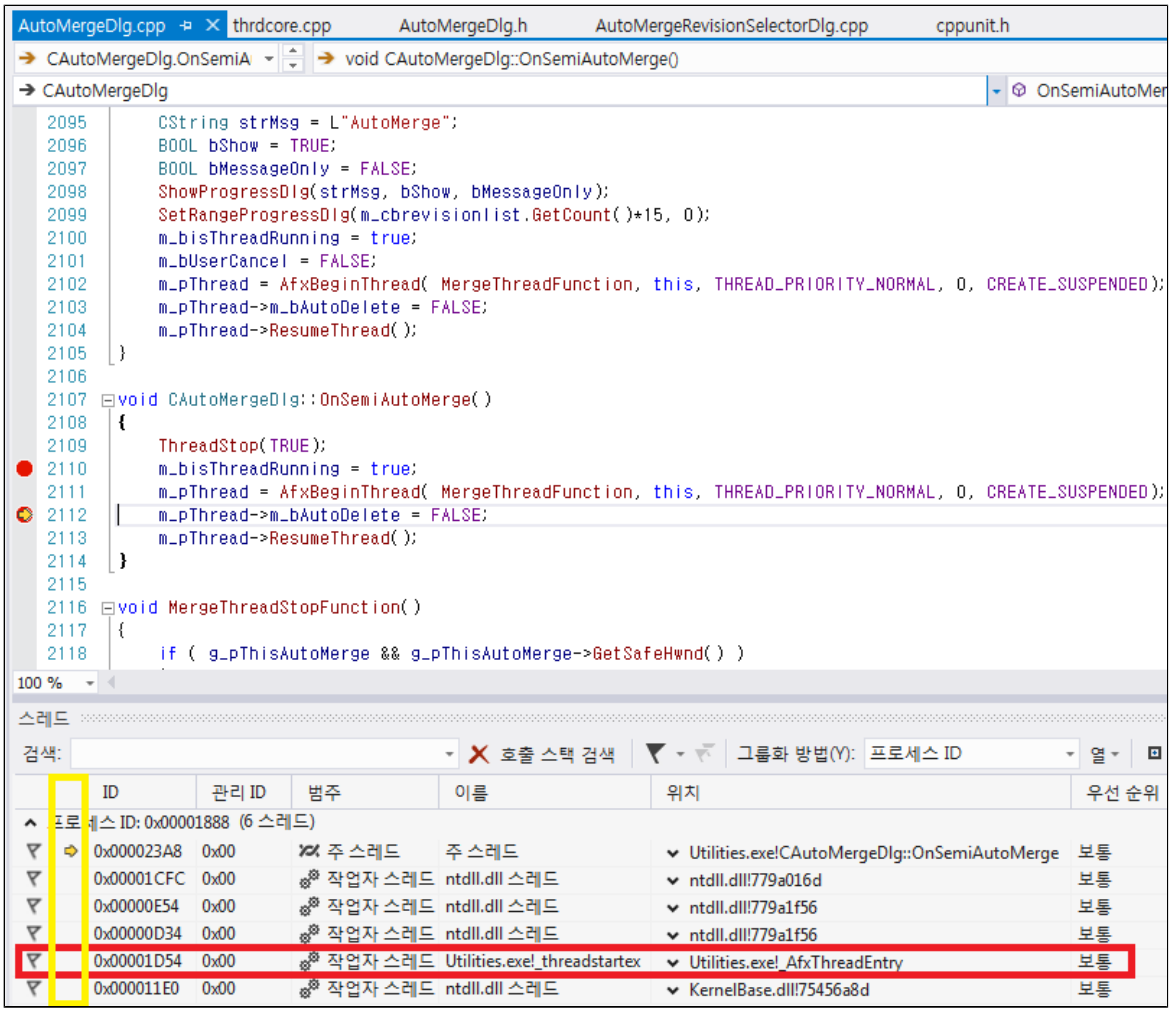
```
UINT MergeThreadFunction(LPVOID lpParam) {
 CAutoMergeDlg* pClass = (CAutoMergeDlg*)lpParam;
 pClass->_OnAutoMerge();
 return 0;
}
```

< AutoMergeTool의 Worker Thread 생성 전/후 디버그창 > => AfxBeginThread() 호출 후 스레드가 생성되고 있음을 볼 수 있다.

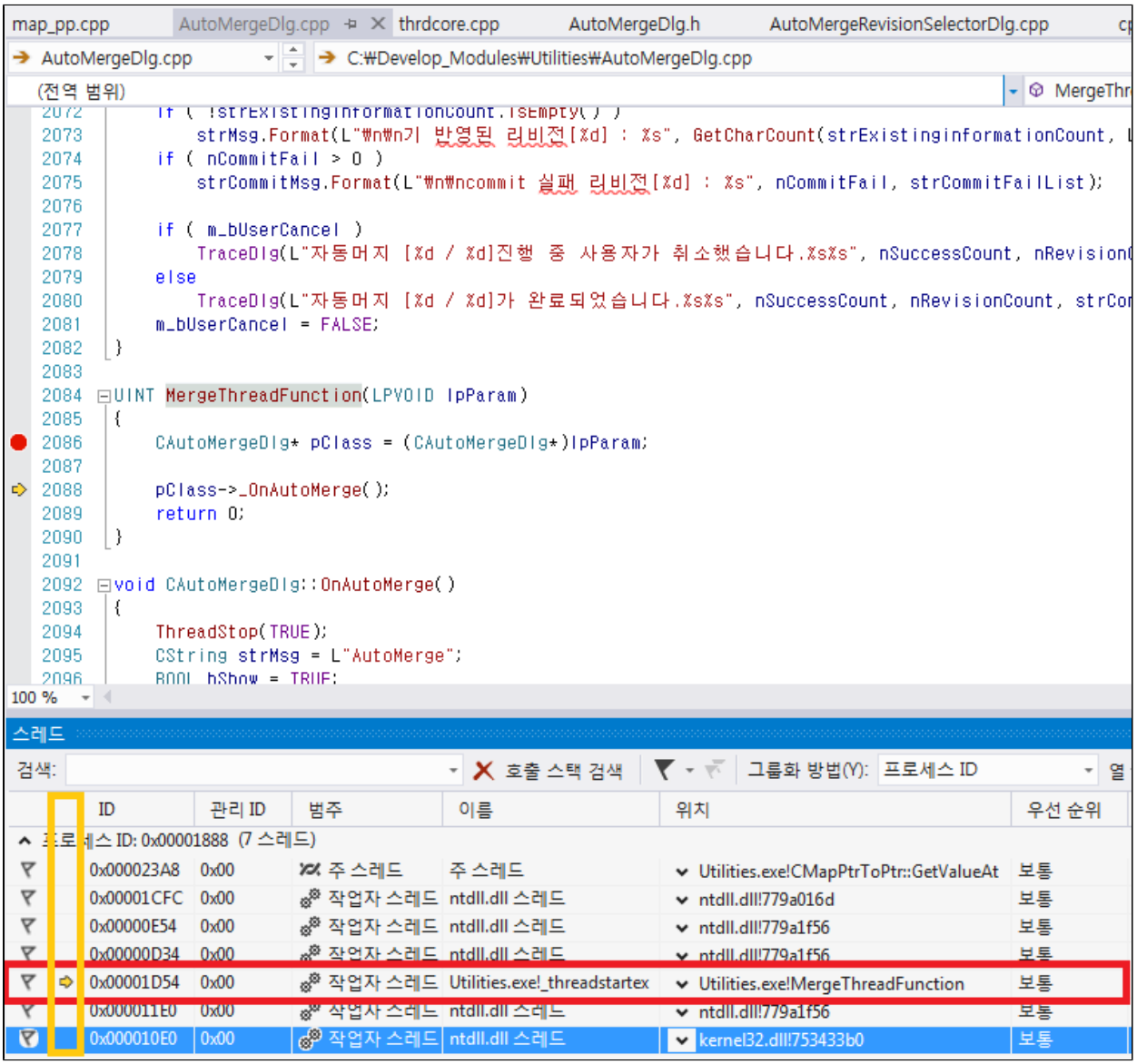
호출 전 :


AutoMergeDlg.cpp
thrdcore.cpp
AutoMergeDlg.h
AutoMergeRevisionSelectorDlg.cpp
cppunit.h
CAutoMergeDlg.OnSemiA
void CAutoMergeDlg::OnSemiAutoMerge()
CAutoMergeDlg
OnSemiAutoMerg
2095 CString strMsg = L"AutoMerge";
2096 BOOL bShow = TRUE;
2097 BOOL bMessageOnly = FALSE;
2098 ShowProgressDlg(strMsg, bShow, bMessageOnly);
2099 SetRangeProgressDlg(m_cbrevisionlist.GetCount()*15, 0);
2100 m_bisThreadRunning = true;
2101 m_bUserCancel = FALSE;
2102 m_pThread = AfxBeginThread(MergeThreadFunction, this, THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
2103 m_pThread->m_bAutoDelete = FALSE;
2104 m_pThread->ResumeThread();
2105 }
2106
2107 void CAutoMergeDlg::OnSemiAutoMerge()
2108 {
2109 ThreadStop(TRUE);
2110 m_bisThreadRunning = true;
2111 m_pThread = AfxBeginThread(MergeThreadFunction, this, THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
2112 m_pThread->m_bAutoDelete = FALSE;
2113 m_pThread->ResumeThread();
2114 }
2115
2116 void MergeThreadStopFunction()
2117 {
2118 if (g_pThisAutoMerge && g_pThisAutoMerge->GetSafeHwnd())
100 %
스레드
검색:
호출 스택 검색
그룹화 방법(Y):
프로세스 ID
열
ID
관리 ID
범주
이름
위치
우선 순위
프로세스 ID: 0x00001888 (4 스레드)
0x000023A8 0x00 주 스레드 주 스레드 Utilities.exe!CAutoMergeDlg::OnSemiAutoMerge 보통
0x00001CFC 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a016d 보통
0x00000E54 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a1f56 보통
0x00000D34 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a1f56 보통

호출 후 :


AutoMergeDlg.cpp
thrdcore.cpp
AutoMergeDlg.h
AutoMergeRevisionSelectorDlg.cpp
cppunit.h
CAutoMergeDlg.OnSemiA
void CAutoMergeDlg::OnSemiAutoMerge()
CAutoMergeDlg
OnSemiAutoMer
2095 CString strMsg = L"AutoMerge";
2096 BOOL bShow = TRUE;
2097 BOOL bMessageOnly = FALSE;
2098 ShowProgressDlg(strMsg, bShow, bMessageOnly);
2099 SetRangeProgressDlg(m_cbrevisionlist.GetCount()*15, 0);
2100 m_bisThreadRunning = true;
2101 m_bUserCancel = FALSE;
2102 m_pThread = AfxBeginThread(MergeThreadFunction, this, THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
2103 m_pThread->m_bAutoDelete = FALSE;
2104 m_pThread->ResumeThread();
2105 }
2106
2107 void CAutoMergeDlg::OnSemiAutoMerge()
2108 {
2109 ThreadStop(TRUE);
2110 m_bisThreadRunning = true;
2111 m_pThread = AfxBeginThread(MergeThreadFunction, this, THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
2112 m_pThread->m_bAutoDelete = FALSE;
2113 m_pThread->ResumeThread();
2114 }
2115
2116 void MergeThreadStopFunction()
2117 {
2118 if (g_pThisAutoMerge && g_pThisAutoMerge->GetSafeHwnd())
100 %
스레드
검색:
호출 스택 검색
그룹화 방법(Y):
프로세스 ID
열
ID
관리 ID
범주
이름
위치
우선 순위
0x000023A8 0x00 주 스레드 주 스레드 Utilities.exe!CAutoMergeDlg::OnSemiAutoMerge 보통
0x00001CFC 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a016d 보통
0x00000E54 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a1f56 보통
0x00000D34 0x00 작업자 스레드 ntdll.dll 스레드 ntdll.dll!779a1f56 보통
0x00001D54 0x00 작업자 스레드 Utilities.exe!_threadstartex Utilities.exe!_AfxThreadEntry 보통
0x000011E0 0x00 작업자 스레드 ntdll.dll 스레드 KernelBase.dll!75456a8d 보통

〈 AutoMergeTool의 작업자스레드 실행 중 디버그창 〉 => MergeThreadFunction()에 진입한 작업자스레드를 확인할 수 있다.



사용자 인터페이스 스레드 (User Interface Thread)

자체의 윈도우와 메시지 루프를 가지고 독립적으로 수행된다. 사용자 인터페이스 스레드를 사용하려면 CWinThread 파생클래스를 만들어 사용한다.

예제) 아래 코드에서 사용자 인터페이스 스레드는 프레임 윈도우를 하나 출력하고, 마우스 버튼이 눌리면 윈도우를 닫는 기능을 한다. (하단 이미지 참고)

UIThread.h

```
class CUIThread : public CWinThread
{
 DECLARE_DYNCREATE(CUIThread)

public:
 virtual BOOL InitInstance();
}
```

UIThread.cpp

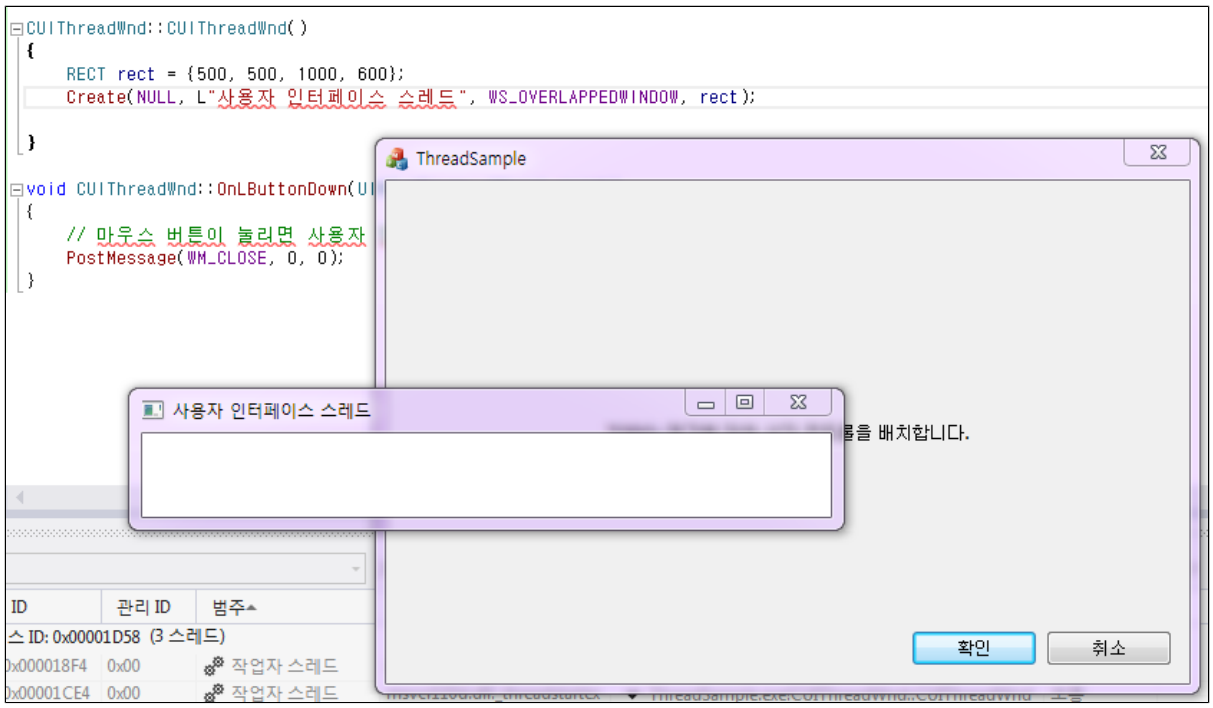
```
#include "UIThread.h"
class CUIThreadWnd : public CFrameWnd
{
public:
 CUIThreadWnd();
protected:
 afx_msg void OnLButtonDown(UINT, CPoint);
 DECLARE_MESSAGE_MAP()
};
IMPLEMENT_DYNCREATE(CUIThread, CWinThread)
BOOL CUIThread::InitInstance()
{
 m_pMainWnd = new CUIThreadWnd();
 m_pMainWnd->ShowWindow(SW_SHOW);
 m_pMainWnd->UpdateWindow();
 return TRUE;
}
BEGIN_MESSAGE_MAP(CUIThreadWnd, CFrameWnd)
 ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
CUIThreadWnd::CUIThreadWnd()
{
 Create(NULL, L"사용자 인터페이스 스레드");
}
void CUIThreadWnd::OnLButtonDown(UINT nFlags, CPoint point)
{
 // 마우스 버튼이 눌리면 사용자 인터페이스 스레드를 종료
 PostMessage(WM_CLOSE, 0, 0);
}
```

ThreadSampleDlg.cpp

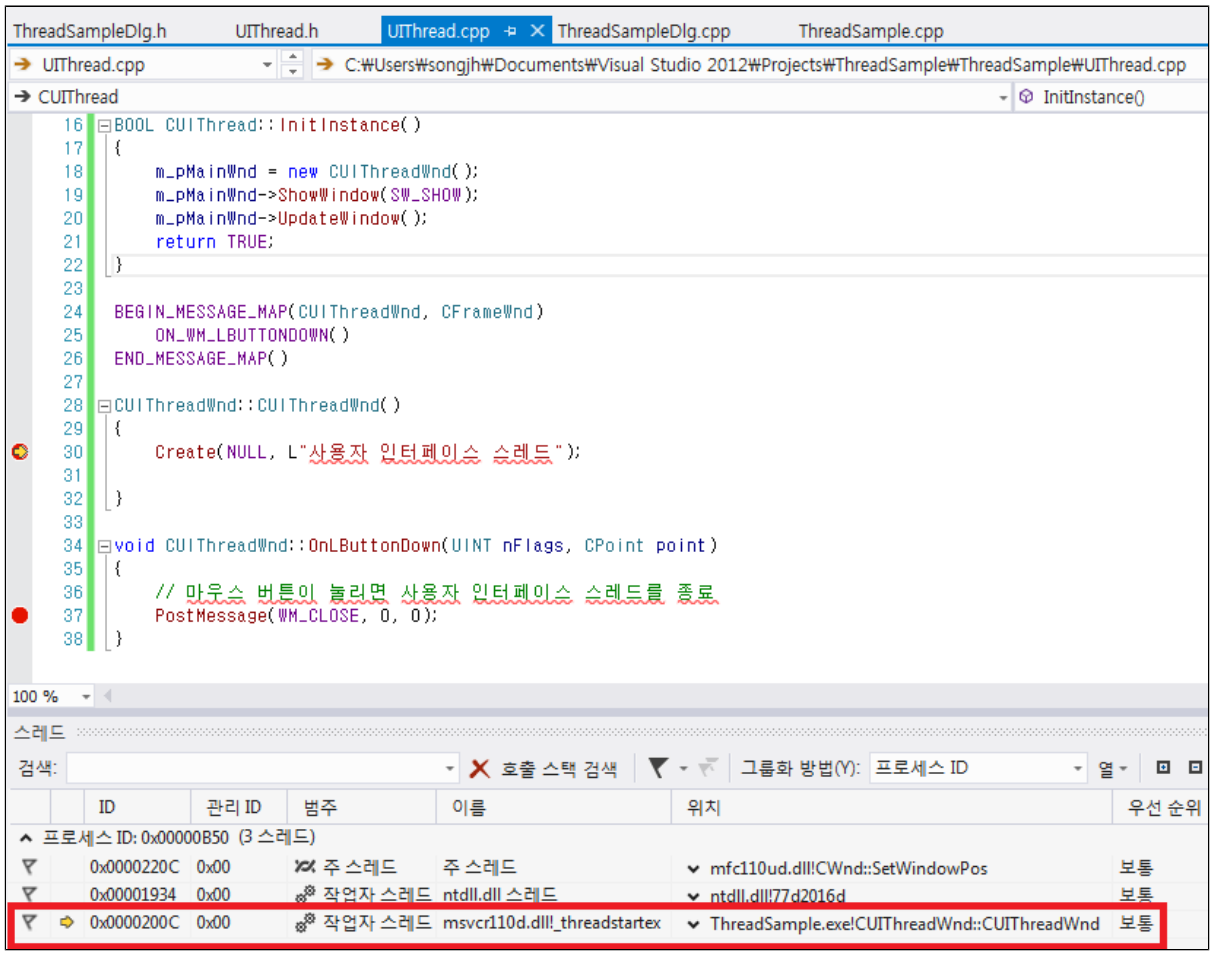
```
BOOL CThreadSampleDlg::OnInitDialog()
{
 CDialogEx::OnInitDialog();
 // 중략
 // TODO: 여기에 추가 초기화 작업을 추가합니다.

 /* AfxBeginThread는 두 가지 형태로 오버로딩되어 있어서,
 *  작업자 스레드/사용자 인터페이스 스레드 구동 시에 모두 사용된다. */
 CWinThread *pThread = AfxBeginThread(RUNTIME_CLASS(CUIThread));
 return TRUE;  // 포커스를 컨트롤에 설정하지 않으면 TRUE를 반환합니다.
}
```

상태	이미지

실행 시



디버깅 시 Thread 탭



작업자 스레드와 사용자 인터페이스 스레드의 차이

	작업자 스레드	사용자 인터페이스 스레드
스레드 구동 방식	백그라운드 작업을 수행할 함수를 만들어 실행시킨다.	CWinThread를 상속받은 클래스를 생성하여 자체 윈도우와 메시지 루프를 사용한다.

메시지 처리 기능	없다.	시스템으로부터 오는 메시지를 받아 처리한다.
스레드 자체 UI 제공 기능	없다.	Dialog를 만들어 사용하면 된다.
주로 사용하는 용도	보통 오래 걸리는 작업이나 무한루프를 수행해야 하는 작업을 할 때 주로 사용된다.	주 프로세스와 독립적으로 수행되는 다중 윈도우를 만들 때 주로 사용된다.

스레드 생성 함수

1. **CreateThread()**
   a. WIN32 api 함수이다.
   b. 스레드 생성함수 호출 시의 리턴값인 핸들로 스레드 조작이 가능하다.
   c. 종료 함수 : ExitThread()
   d. 참고 MSDN : https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms682453(v=vs.85).aspx
2. **_beginthread()**
   a. c 라이브러리 함수이다.
   b. 스레드 생성함수 호출 시의 리턴값인 핸들로 스레드 조작이 가능하다.
   c. 종료 함수 : endthread() 함수
   d. 참고 MSDN : https://msdn.microsoft.com/ko-kr/library/7t9ha0zh.aspx
3. **AfxBeginThread()**
   a. 가장 많이 사용하는 스레드 생성 함수
   b. 스레드 생성함수 호출 시 리턴값은 CWinThread* 형인데, 이 객체를 통해 스레드 조작이 가능하다.
   c. 종료 함수 : AfxEndThread() -> 스레드 내부에 메모리를 동적할당(new,malloc)해놓고 delete를 안해서 메모리 릭이 날 염려가 있다. 자세한 건 다음주 세미나에
   d. 참고 MSDN : https://msdn.microsoft.com/ko-kr/library/s3w9x78e.aspx

구문

```
// 작업자 스레드 생성 함수
CWinThread* AfxBeginThread(
   AFX_THREADPROC pfnThreadProc,
   LPVOID pParam,
   int nPriority = THREAD_PRIORITY_NORMAL,
   UINT nStackSize = 0,
   DWORD dwCreateFlags = 0,
   LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL
);
// 유저 인터페이스 스레드 생성 함수
CWinThread* AfxBeginThread(
   CRuntimeClass* pThreadClass,
   int nPriority = THREAD_PRIORITY_NORMAL,
   UINT nStackSize = 0,
   DWORD dwCreateFlags = 0,
   LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL
);
```

매개 변수	설명
AFX_THREADPROC pfnThreadProc	작업자 스레드에 대한 제어 함수를 가리킵니다. **NULL**이 될 수 없습니다. 이 함수는 다음과 같이 선언해야 합니다. UINT __cdecl MyControllingFunction(LPVOID pParam);
CRuntimeClass* pThreadClass	CWinThread에서 파생된 개체의 RUNTIME_CLASS.
LPVOID pParam	pfnThreadProc의 함수 선언에 매개 변수가 전달되는 것처럼 기능 제어에 매개 변수가 전달됩니다. **-> worker thread에서만 유효**
int nPriority	스레드의 원하는 우선 순위. 사용 가능한 우선 순위의 전체 목록과 설명을 보려면 Windows SDK에서 SetThreadPriority를 참조하십시오.

UINT nStackSize	새 스레드의 스택 크기를 지정합니다(바이트 단위). 0인 경우 스택 크기 기본값은 만드는 스레드와 스택의 크기가 동일합니다.
DWORD dwCreateFlags	스레드 생성을 제어하는 추가 플래그를 지정합니다. 이 플래그는 두 값 중 하나를 포함할 수 있습니다. i. **CREATE_SUSPENDED** 일시 중단 횟수 1로 스레드를 시작합니다. 스레드가 실행되기 전에 **CWinThread** 개체의 모든 멤버 데이터를 초기화하려면 (예: m_bAutoDelete 또는 파생 클래스의 모든 멤버) **CREATE_SUSPENDED**를 사용합니다. 초기화가 완료되면CWinThread::ResumeThread를 사용하여 스레드 실행을 시작합니다. 스레드는 **CWinThread::ResumeThread**가 호출될 때까지 실행되지 않습니다. ii. **0** 스레드를 만든 후 즉시 시작합니다.
LPSECURITY_ATTRIBUTES lpSecurityAttrs	스레드의 보안 특성을 지정하는 SECURITY_ATTRIBUTES를 가리킵니다. **NULL**인 경우 스레드 생성과 동일한 보안 특성이 사용됩니다. 이 구조에 대한 자세한 내용은 Windows SDK를 참조하십시오.

4. MFC에서 스레드 생성하기

MFC에서 스레드를 만드는 방법은 두 가지가 있다.

a. CWinThread의 멤버함수인 CreateThread()를 사용한다.

i. Win32 api의 CreateThread() 함수가 아닌 CWinThread 멤버함수 CreateThread() 이다.

CWinThread::CreateThread()는 Win32 api CreateThread()와는 달리 MFC에서 사용되기 위한 부분이 추가되었다.

b. AfxBeginThread()를 사용한다. -> CWinThread 객체 생성이 가능하다.

i. 작업자 스레드/UI 스레드 생성 시 사용할 수 있으며, 각각의 함수가 정의되어 있다.

ii. 내부에서 CWinThread::CreateThread()를 호출한다.

출처

VISUAL C++ 6 완벽 가이드 / 김용성 저

https://msdn.microsoft.com/ko-kr/library/s3w9x78e.aspx

https://msdn.microsoft.com/ko-kr/library/7t9ha0zh.aspx

https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms682453(v=vs.85).aspx

http://blog.naver.com/illusyon/8305301

http://torpedo7.egloos.com/m/79879