MFC 멀티스레드 프로그래밍 [2016.07.14]

스레드 실행 제어

MSDN AfxBeginThread() : https://msdn.microsoft.com/ko-kr/library/s3w9x78e.aspx - 이 함수를 호출하여 새 스레드를 만든다.

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);
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. 1. **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. 2. **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.

1. 스레드 멈추기 진행시키기

a. dwCreateFlags에 0을 넘겨주면 스레드 생성되는 동시에 실행이 되고, CREATE_SUSPENDED를 인자로 넘겨주면 스레드가 멈춰진 상태로 생성된다.
b. 스레드가 실행되고 있는 동안에 스레드 내부에서 ::SuspendThread 함수를 호출하면 스레드 동작을 멈출 수 있다.
c. 멈추어진 스레드를 다시 동작시키려면 다음 예시와 같이 ::AfxBeginThread 함수의 반환값으로 얻어진 CWinThread 오브젝트의 포인터를 이용하여 CWinThread::ResumeThread를 호출하면 된다.
d. 예시 - AutoMerge Tool

AutoMergeDlg.cpp

```
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;
}
```

2. 스레드의 동작을 잠시 쉬게 하기

a. 스레드 동작을 잠시 쉬게 하려면 스레드 내부에서 ::Sleep 함수를 호출하면 된다.

```
/* 스레드의 동작을 1초 동안 멈추기
1초 동안 해당 스레드가 CPU를 전혀 사용하지 않기 때문에 다른 프로세스나 스레드가 CPU를 사용할 수
있다. */
::Sleep(1000);

/* Sleep(0)을 호출하면
CPU를 사용하려고 기다리고 있는 우선순위가 높거나 같은 프로세스나 스레드에게 CPU 사용권이
넘어간다.
-> 이런 프로세스나 스레드가 없을 경우 아무 일도 일어나지 않는다. */
::Sleep(0);
```

3. 스레드 종료하기

a. **사용자 인터페이스 스레드**
   i. 자체적으로 메시지 큐를 가지고 있기 때문에 WM_QUIT 메시지를 보내주면 스레드가 종료된다.

```
m_pThread->PostThreadMessagge(WM_QUIT, 0, 0);
```

b. **작업자 스레드**
   i. 주 프로세스에서 :: TerminateThread 함수를 호출하여 스레드를 강제 종료 시킬 수 있지만 바람직한 방법이 아니다.
   -> 메모리 해제 등 정리 작업을 못하기 때문
   ii. 외부에서는 플래그만 설정하여 스레드에게 종료되어야 한다는 사실을 알려주고, 스레드가 자체적으로 실행을 마치는 것이 좋다.

iii.
```
void BeginThread()
{
 static BOOL bContinue = TRUE;
 CWinThread *pThread = ::AfxBeginThread(ThreadFunc,bContinue);
}

UINT ThreadFunc(LPVOID pParam)
{
 BOOL *pContinue = (BOOL*) pParam;
 while(*pContinue)
 {
  // 스레드가 실행된다.
 }
 return 0;
}

void StopThread()
{
/* 스레드는 주 프로세스와 독립적으로 수행되기 때문에
주 프로세스에서 플래그를 설정하자마자 스레드가 바로 종료되지 않는다.
주 프로세스가 CPU를 다 쓰고 나서, 스레드가 CPU를 점유하게 된 후
스레드 함수의 나머지 부분을 수행 후 종료된다. */
 bContinue = FALSE;

/* 스레드 종료를 위한 플래그를 설정한 후 스레드가 완전히 종료된 것을
확실히 확인하고 그 후에 어떤 작업을 하고 싶으면 다음과 같이 하면 된다. */
 DWORD dwRetCode = ::WaitForSingleObject(pThread->m_hThread, 2000);
 // 위 함수를 호출하면 pThread가 가르키는 스레드가 종료될 때까지 프로세스의 실행을
멈춰준다.
 // 함수 두 번째 인자로 얼마 동안 기다릴 것인지를 설정할 수 있다. (INFINITE:무한정)

 if(dwRetCode == WAIT_OBJECT_0)
 {
  // 스레드가 종료된 후 해야 할 작업들
 }
 else if(dwRetCode == WAIT_TIMEOUT)
 {
  // 2초 동안 스레드가 종료되지 않았을 때 해야 할 에러 처리 작업
 }
}
```

실행의 우선순위

1. 우선순위에 따른 CPU의 사용

a. CPU는 한번에 한 가지 작업밖에 할 수 없다. 멀티스레드 APP을 실행할 때는 여러 개의 스레드를 짧은 시간 동안 돌아가면서 실행해 줌으로써 동시에 실행되는 것처럼 보이게 하는 것이다.
b. 어떤 스레드에 CPU 사용권을 줄 것인지는 운영체제의 스케줄러가 정한다.
c. 스케줄러는 각 스레드마다 설정된 우선순위에 의하여 우선순위가 높은 스레드가 우선적으로 CPU를 사용할 권한을 부여한다.

2. 스레드의 우선순위 설정

a. 스레드의 우선순위 설정 방법
    i. 스레드를 생성할 때, AfxBeginThread 함수의 nPriority를 이용하여 설정할 수 있다.
    ii. 스레드가 생성된 후 CWinThread::SetThreadPriority 함수를 이용하여 설정할 수 있다.
b. 스레드의 우선순위는 스레드를 생성한 주 프로세스의 우선순위에 대한 상대적인 값으로 설정되며, 기본적으로는 주 프로세스와 같은 우선순위를 갖는 THREAD_PRIORITY_NORMAL로 설정된다.
c. 표) 스레드의 우선 순위

우선순위	의미
THREAD_PRIORITY_IDLE	프로세스가 REALTIME_PRIORITY_CLASS 우선순위를 가질 때는 우선순위 16을 갖고 그 외의 경우에는 우선순위 1을 갖음

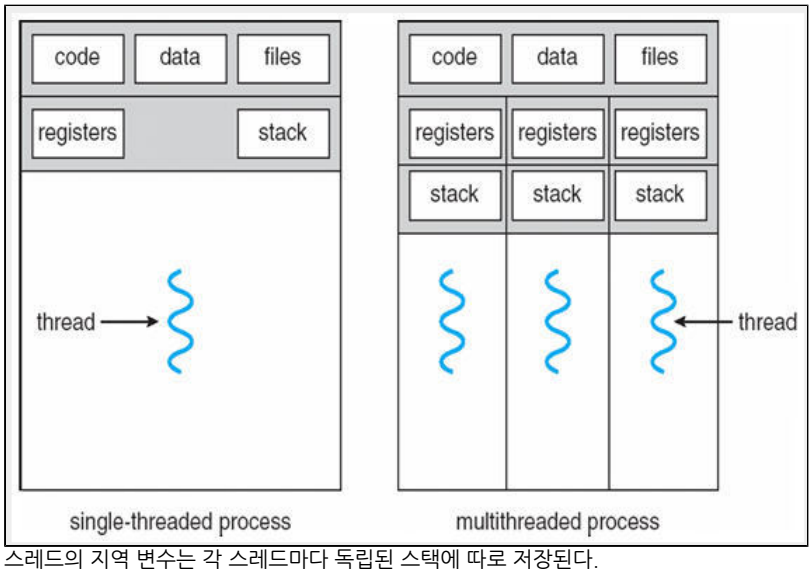
THREAD_PRIORITY_LOWEST	프로세스의 우선순위보다 2단계 낮은 우선순위를 갖음
THREAD_PRIORITY_BELOW_NORMAL	프로세스의 우선순위보다 1단계 낮은 우선순위를 갖음
THREAD_PRIORITY_NORMAL	프로세스의 우선순위와 같은 우선순위를 갖음
THREAD_PRIORITY_ABOVE_NORMAL	프로세스의 우선순위보다 1단계 높은 우선순위를 갖음
THREAD_PRIORITY_HIGHEST	프로세스의 우선순위보다 2단계 높은 우선순위를 갖음
THREAD_PRIORITY_CRITICAL	프로세스가 REALTIME_PRIORITY_CLASS 우선순위를 가질 때는 우선순위 31을 갖고 그 외의 경우에는 우선순위 16을 갖음

.

3. ### 스케줄러의 우선순위 변경
    a. 우선순위가 낮아 너무 오랫동안 CPU를 사용하지 못하고 기다리고 있는 프로세스나 스레드가 있으면 우선순위를 조금 높여 CPU를 사용할 수 있도록 해 준다.
    b. 입력 포커스를 받거나 활성화된 프로세스나 스레드는 우선순위를 높여줌으로써 사용자의 입력에 우선적으로 반응할 수 있도록 해 준다.

스레드와 메모리

1. ### 주 프로세스의 전역 변수 참조
    a. 스레드는 주 프로세스의 전역 변수를 참조할 수 있다. 따라서 주 프로세스와 스레드 간의 상호 작용을 하는 가장 간단한 방법은 전역 변수를 만들어 두고 주 프로세스와 스레드가 이를 공유하는 것이다.

2. ### 스레드의 지역 변수
    a. 

    b. 스레드의 지역 변수는 각 스레드마다 독립된 스택에 따로 저장된다.
    c. **예시)**
        i. 하나의 작업자 스레드 함수를 이용하여 두 개의 스레드를 구동시킴

```
AfxBeginThread(ThreadFunc, NULL);
AfxBeginThread(ThreadFunc, NULL);
```

```
UINT ThreadFunc(LPVOID pParam)
{
 int nCount;
 while(1)
 {
  nCount++;
 }
}
```

ii. 각각의 스레드가 독립된 스택을 만들고 지역 변수를 각각 따로 저장하기 때문에 두 개의 스레드는 독립적으로 실행되며, nCount 변수도 각각 다른 값을 가지고 있게 된다.

Sample - 4 Core환경에서 8 Thread 구동 예제

1. 목적

a. 스레드 실행제어에 대해 확인할 수 있다.
   i. 스레드 실행
   ii. 스레드 중지(Suspend)
   iii. 스레드 정상종료
b. 멀티 스레드 환경에서의 스레드 우선순위를 확인할 수 있다.
c. 디버깅 시 스레드는 각각 독립된 스택을 가지고 있음을 확인할 수 있다.

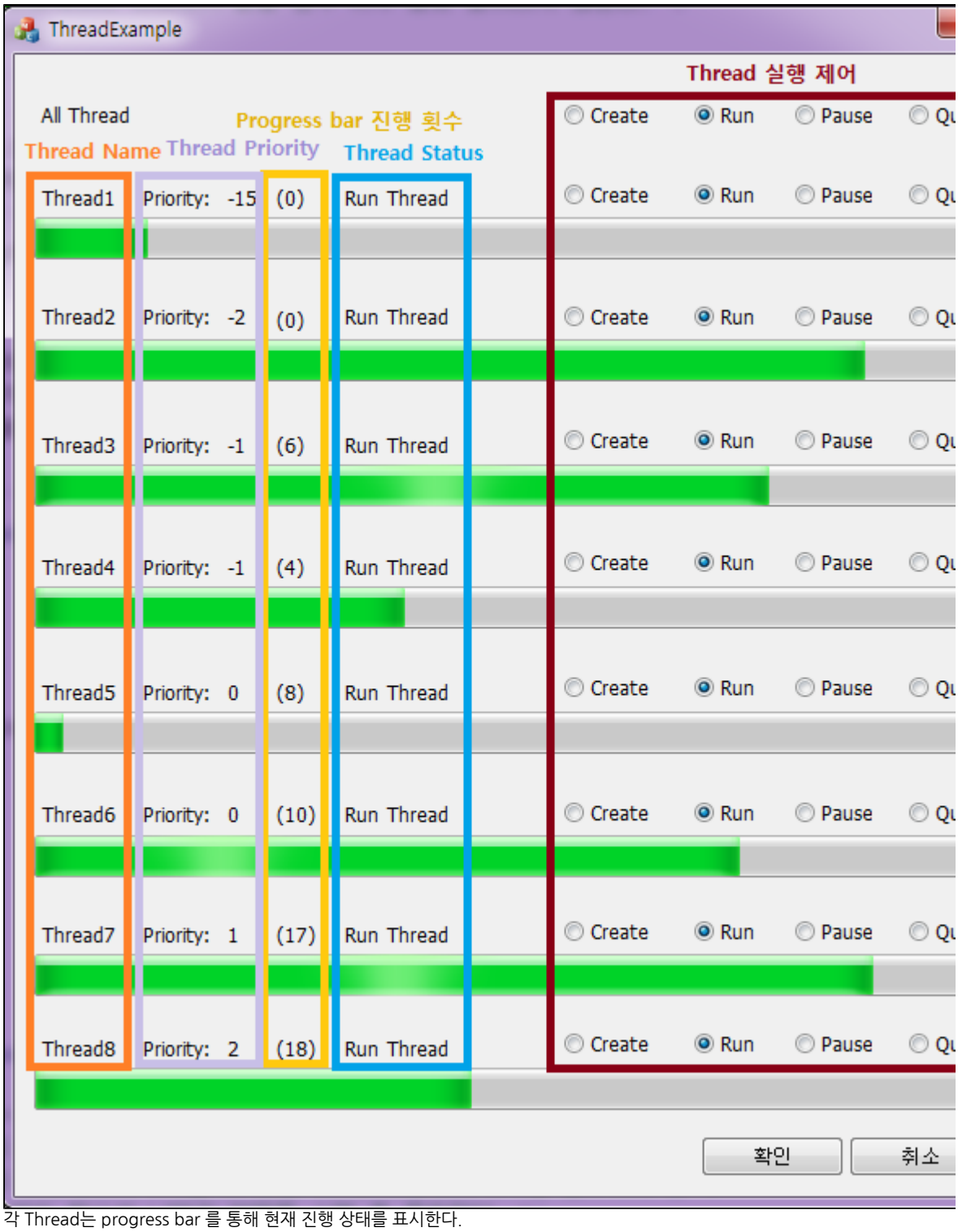
2. Sample Source SVN

a. https://tr00129.infraware.net/svn/PolarisOffice7_Engine/POTester/ThreadSample

3. 구동 방법

a. ThreadSample.sln 실행 후 ThreadExample을 시작프로젝트로 설정하여 구동

4. Sample 설명



a.
b. 각 Thread는 progress bar 를 통해 현재 진행 상태를 표시한다.
c. 각 Thread들은 Class의 멤버 변수로 관리된다. (CWinThread*)

ThreadExample.h

```
CWinThread *m_pThread[8]; // 각 thread를 제어하기 위한 변수
```

d. 각 Thread들은 임의로 priority가 적용되어 있으며, 코드에서 nThreadPriority 변수를 수정하여 priority를 조정할 수 있다.

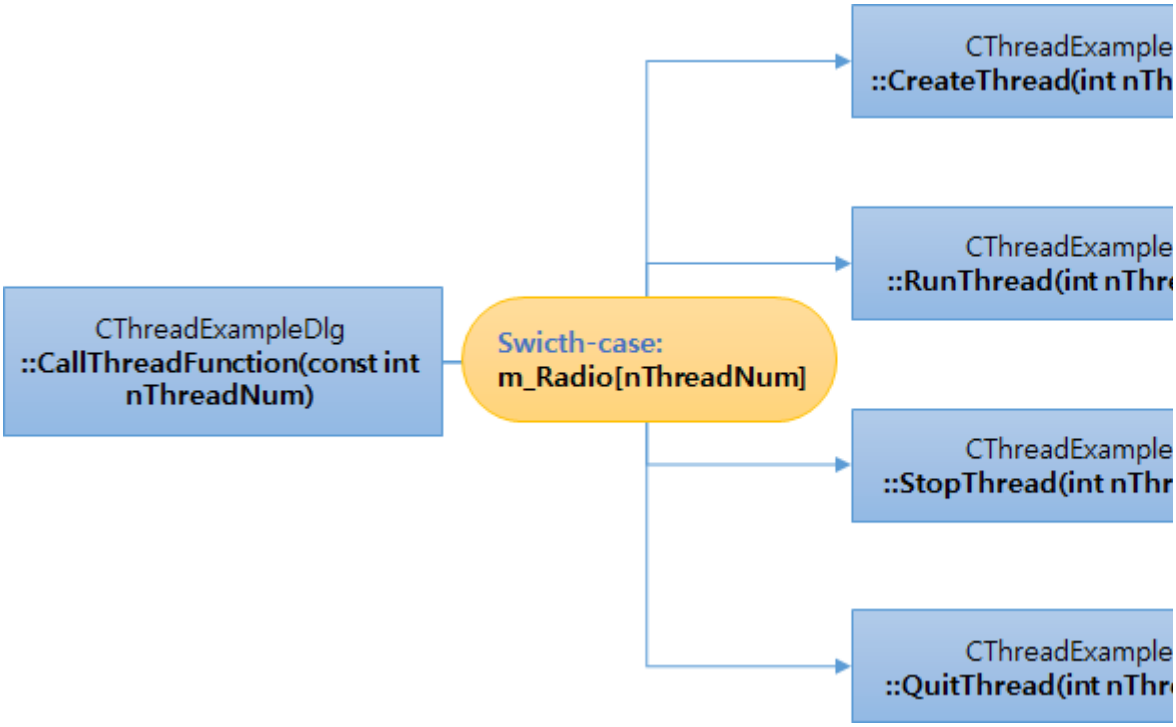
ThreadExample.cpp

```
// 각 thread들의 우선순위
static int nThreadPriority[8] = {THREAD_PRIORITY_IDLE, THREAD_PRIORITY_LOWEST,
THREAD_PRIORITY_BELOW_NORMAL, THREAD_PRIORITY_BELOW_NORMAL,
 THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_NORMAL,
THREAD_PRIORITY_ABOVE_NORMAL, THREAD_PRIORITY_HIGHEST};
```

e. 각 Thread들은 실행 제어 radio button을 통해 구동된다.
   i. Create -> Run 순으로 진행해야 하며, 최초 구동시에는 자동으로 Thread Create 과정이 진행된다.
   ii. Pause -> Run 순으로 진행할 경우 정지했던 시점부터 다시 Thread가 구동된다.
   iii. Quit 후 Thread를 실행하고 싶으면 Create->Run순으로 실행해야 한다. (Quit 시에는 Thread를 정상 종료한다.)

5. 내부 구동 방식

a.



6. 코딩 중의 Thread 관련 이슈

a. UINT ThreadFunction() 함수 내부에서 **Progressbar를 그릴 경우 progress의 value는 max에 도달해도 progress bar가 max까지 그려지지 않는 문제**
   i. **원인** : UI가 progress bar를 끝까지 그릴 시간 없이 MAX값에서 바로 0으로 이동 후 0을 그리도록 호출하기 때문에 발생한 문제
   ii. **해결 방안**
      1. Progressbar value는 Max값보다 크게 하여 UI가 progress bar를 그릴 시간을 줌
         -> 아래 코드에서 if(*(m_nProgressValue) > PROGRESS_MAX +10) 부분에 해당됨.
      2. progress bar를 그리는 루틴을 Timer로 이동시켜 Thread와는 별도로 동작하도록 수정

ThreadExample.cpp

```
// (수정 후 코드) 각 Thread들이 실행하는 함수
UINT ThreadFunction(LPVOID lpParam)
{
 while(*(m_bProgress))
 {
  //////////////시간을 보내기 위한 무의미한 작업//////////////////////
  // Progress bar 위치 이동
  (*(m_nProgressValue))++;
  if(*(m_nProgressValue) > PROGRESS_MAX +10)
  {
   (*(m_nProgressTotalCompleteCount))++;
  }
 return 0;
 }
}
```

b. UINT ThreadFunction() 함수 내부에서 **Sleep(100)을 호출하여 Thread간의 우선순위가 차이 없는 문제**
   i. **Sleep(100) 호출 이유** : progress bar의 진행속도를 늦추기 위해 thread를 강제로 잠시 쉬게 함.
   ii. **우선순위 차이가 없었던 이유** : 열심히 일해야 OS가 우선순위에 따라 CPU를 사용할 수 있도록 스케줄링 하는데 실질적으로 일하는 시간보다 Sleep으로 양보하는 시간이 더 길기 때문.

ThreadExample.cpp

```
// (수정 전 코드) 각 Thread들이 실행하는 함수
UINT ThreadFunction(LPVOID lpParam)
{
 while(*(m_bProgress))
 {
  Sleep(100);
  // Progress bar 위치 이동
 }
 return 0;
}
```

c. UINT ThreadFunction() 함수 내부에서의 **malloc() 호출 코드로 인해 "All Thread"를 "Pause"할 경우 프로그램 자체가 lock 걸리는 문제**
   i. **malloc() 호출 이유** : progress bar의 진행속도를 늦추기 위해 sleep(100)제거 후 시간이 걸리는 작업인 malloc()을 추가함.
   ii. **thread 중지 시 Lock 걸린 이유** : malloc() 함수가 thread safe하기 때문에 suspendThread()를 호출할 경우 malloc() 함수 내부에서 lock이 걸리는 문제가 발생함. - 메모리를 할당하는 부분이어서 동시에 thread 한 개만 접근 가능하기 때문
   iii. **해결 방안** : malloc() 호출할 필요가 없어서 제거함

ThreadExample.cpp

```
// (수정 전 코드) 각 Thread들이 실행하는 함수
UINT ThreadFunction(LPVOID lpParam)
{
 while(*(m_bProgress))
 {
  //////////////시간을 보내기 위한 무의미한 작업//////////////////////
  int nCount = 0;
  int tmp;
  while(nCount++ <10)
  {
   //Sleep(100);
   //int *tmp = (int*)malloc(sizeof(int)*1000*1000);
   for(int i=0; i<1000; i++)
   {
    for(int j=0; j<1000; j++)
     tmp = (i+1)*(j+1)*(i+2)*(j+3)*(j+4)/(i+j+1)*(i+1);
   }
   //free(tmp);
  }
  ////////////////////////////////////////////////////////////////
  // Progress bar 위치 이동
 }
 return 0;
}
```

d. 메모리 릭 이슈
   i. malloc() 후 제대로 해제하지 않아 발생 -> malloc()은 필요 없는 코드라서 제거함.
   ii. 윈도우 확인/취소 버튼 누를 때 OnDestory() 함수가 제대로 호출되지 않아 발생 -> Message Map에 ON_WM_DESTROY() 추가함.
   iii. 스레드 강제종료 함수 파라미터 잘못 사용하여 스레드가 종료되지 않음 -> 파라미터 수정하여 정상동작 확인함 - 아래 코드
   iv. 스레드 종료되었는지 확인하는 함수 파라미터 잘못 사용하여 정상 구동되지 않음 -> 파라미터 수정하여 정상동작 확인함 - 아래 코드

ThreadExample.cpp

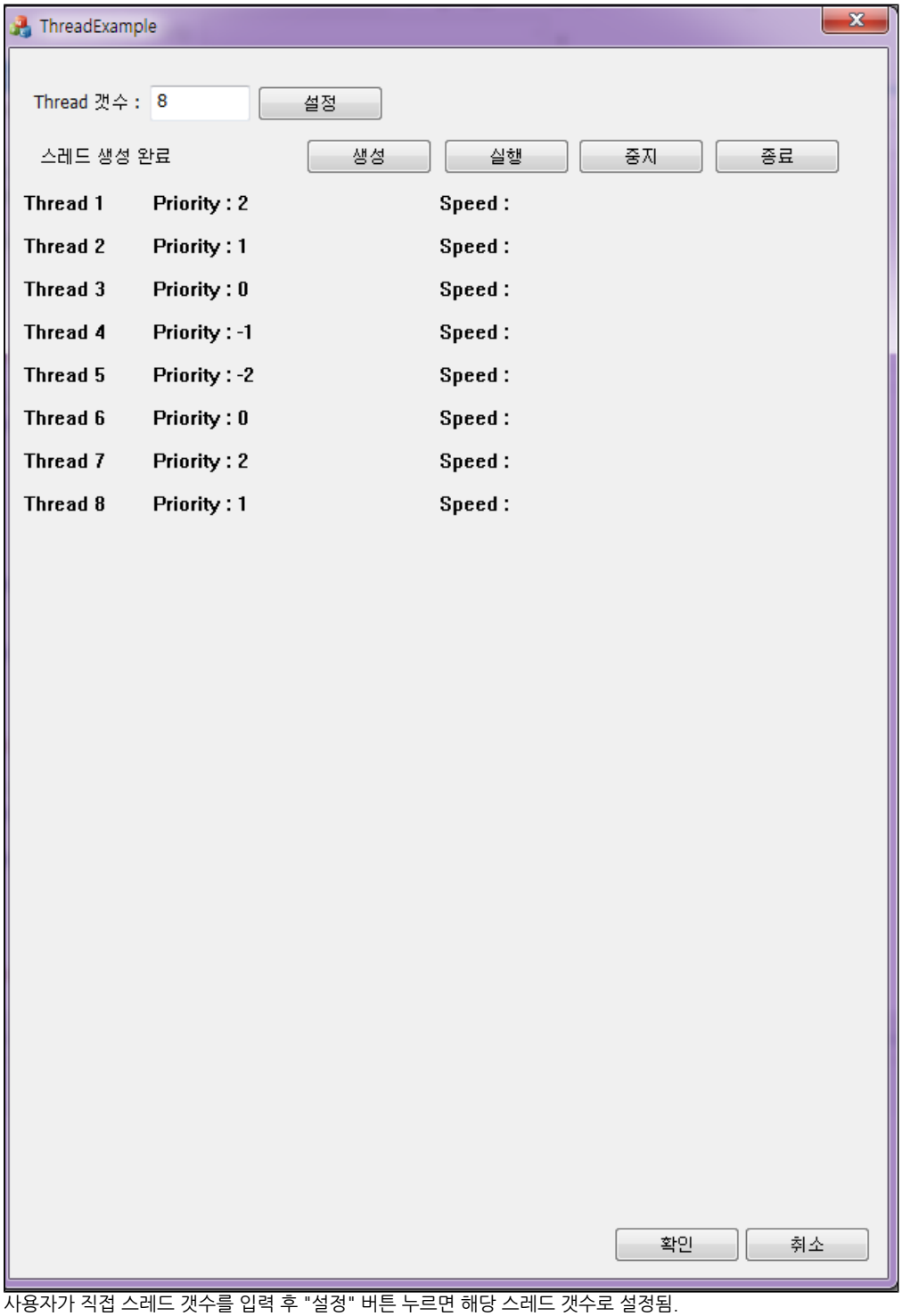
```
DWORD dwExitCode=0;
 ::GetExitCodeThread(m_pThread[nThreadNum]->m_hThread, &dwExitCode);
 if ( dwExitCode != 0 )
 {
  DWORD dwResult = ::WaitForSingleObject(m_pThread[nThreadNum]->m_hThread,
1000); // (4)번 이슈
  if(dwResult == WAIT_TIMEOUT)
  {
   ::TerminateThread(m_pThread[nThreadNum]->m_hThread, 0); // (3)번 이슈
   SetDlgItemText(IDC_STATIC_STATUS, L"강제 종료했습니다.");
  }
  else if(dwResult == WAIT_OBJECT_0)
   SetDlgItemText(IDC_STATIC_STATUS, L"종료되었습니다.");
 }
```

WaitForSingleObject MSDN : https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms687032(v=vs.85).aspx
GetExitCodeThread MSDN : https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms683190(v=vs.85).aspx

   v. 이 외 누락된 delete 코드들 추가

Sample2

1. **목적**
   a. 각 사용자의 pc 마다 core 갯수가 달라 위 sample로 우선순위 확인이 어려우므로 **직접 스레드 갯수를 입력받아 구동하도록 만듬**
2. **구동 방법**
   a. SampleExample2 를 시작프로젝트로 설정하여 구동
3. **Sample2 UI**



   a.
   b. 사용자가 직접 스레드 갯수를 입력 후 "설정" 버튼 누르면 해당 스레드 갯수로 설정됨.
   c. 그 후 "생성"->"실행" 을 순서대로 구동하여 스레드를 동작시킴
   d. 동작중인 스레드의 속도는 "Speed:" 에서 확인 가능함
   e. Sample1과는 다르게 모든 스레드가 한번에 생성/실행/중지/종료 됨. (따로 스레드를 선택하여 제어하는 기능 없음)