```
// Vahe Karamian - Windows 32 Multithreaded Programming
// Filename: multithreadedHelloWorld.cpp

// Multithereaded Hello World Program

#include <windows.h>
#include <stdio.h>

// Prototype for Hello application worker thread procedure
DWORD WINAPI WorkerThreadProc( LPVOID lpThreadParameter );

// Main program entry point
int main( void )
{
  HANDLE  hWorkerThread;
  DWORD   dwWorkerThreadId;
  DWORD   dwPrimaryThreadId;

  // Determine and save the primary thread's identifier
  dwPrimaryThreadId = GetCurrentThreadId( );

  printf( "[%08x] Multithreaded hello application started.\n", dwPrimaryThreadId );

  // Launch the worker thread
  hWorkerThread =
    CreateThread( NULL,             // Default security attributes
                  0,                // Default stack size
                  WorkerThreadProc, // Thread proc to start at
                  (LPVOID)3,        // Argument: work for 3 seconds
                  0,                // No special flags
                  &dwWorkerThreadId // Out: thread ID of worker
                );

  if( hWorkerThread != NULL )
  {
    printf( "[%08x] Worker thread ID = 0x%08x.\n", dwPrimaryThreadId, dwWorkerThreadId );

    // Give the worker thread time to do its thing
    printf( "I am going to sleep for 6 seconds - starting Worker Thread!\n" );
    Sleep( 6000 );
    printf( "I am back - terminating Worker Thread!\n" );

    // Read the worker thread's exit code and cleanup
    DWORD dwThreadExitCode;

    GetExitCodeThread( hWorkerThread, &dwThreadExitCode );

    printf( "[%08x] Worker thread exit code = 0x%08x.\n", dwPrimaryThreadId, dwThreadExitCode );

    CloseHandle( hWorkerThread );
  }
  else
  {
    printf( "[%08x] Failed to create worker thread: error 0x%x\n", dwPrimaryThreadId, GetLastError( ) );
  }

  printf( "[%08x] Multithreaded hello application exiting.\n", dwPrimaryThreadId );

  return( 0 );
}


// Thread entry procedure for the worker thread
DWORD WINAPI WorkerThreadProc( LPVOID lpThreadParameter )
{
  // For this thread, lpThreadParameter is used to indicate the number
  // of seconds this thread should execute for.
  DWORD dwNumSeconds = (DWORD) lpThreadParameter;
  DWORD dwThreadId = GetCurrentThreadId( );

  while( dwNumSeconds-- > 0 )
  {
    Sleep( 1000 );

    printf( "[%08x] Hello, multithreaded world!\n", dwThreadId );
  }

  // Return from thread procedure and specify an exit code equal to our thread ID.
  return( dwThreadId );
```

```
}
```