# Creazione di un processo in Windows

**// Funzione delle API di Windows**

**// Crea un nuovo processo e il relativo thread primario.**

// Il nuovo processo viene eseguito nel contesto di protezione
// del processo chiamante

```
BOOL WINAPI CreateProcess(
    _In_opt_      LPCTSTR  lpApplicationName,
    _Inout_opt_   LPTSTR   lpCommandLine,
    _In_opt_      LPSECURITY_ATTRIBUTES   lpProcessAttributes,
    _In_opt_      LPSECURITY_ATTRIBUTES   lpThreadAttributes,
    _In_          BOOL  bInheritHandles,
    _In_          DWORD  dwCreationFlags,
    _In_opt_      LPVOID  lpEnvironment,
    _In_opt_      LPCTSTR  lpCurrentDirectory,
    _In_          LPSTARTUPINFO  lpStartupInfo,
    _Out_         LPPROCESS_INFORMATION  lpProcessInformation);
```

http://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx

# SAL - Source-code Annotation Language

È un linguaggio che fornisce una serie di **annotazioni** per descrivere come una funzione utilizza i parametri.

Il file intestazione <sal.h> definisce le annotazioni.
È utilizzato in Visual C ++ in Visual Studio 2012, per ridurre i difetti del codice C / C++.

Utilizzare il linguaggio SAL è un modo conveniente per consentire al compilatore di controllare il codice *automaticamente* e poter rilevare degli errori.

Per i **puntatori** le annotazioni possono essere ad esempio:

| | |
|---|---|
| _In_ | _In_opt_ |
| _Out_ | _Out_opt_ |
| _Inout_ | _Inout_opt_ |

```
// Incorrect
void Func1(_In_ int *p1)
{
    if (p1 == NULL)
        return;

    *p1 = 1;
}

// Correct
void Func2(_Inout_ PCHAR p1)
{
    if (p1 == NULL)
        return;

    *p1 = 1;
}
```

# Programma in C++ che visualizza il proprio PID

```
//   ProcessoFiglio.cpp

#include <iostream>
#include <windows.h>
using namespace std;

int main()
{
 // visualizza il Process ID assegnato dal sistema
 cout <<"\n =======> Inizia l'esecuzione del processo con PID: "<<GetCurrentProcessId( )<<endl;
 cout <<"\n          Sono il processo FIGLIO"<<endl;
 cout<<" ================== Fine esecuzione processo con PID: " <<GetCurrentProcessId()<<endl;
 cout<<endl;
system ("pause");
}
```

# Programma in C++ che crea un processo figlio

```
// ProcessoPadre.cpp
//Creazione di un processo con una funzione di Windows
//   (API-Application Programming Interface )
// Consente di creare un processo figlio  e il nuovo processo creato è indipendente dal
// processo padre  pur  esistendo una relazione padre-figlio.

#include <iostream>
#include <windows.h>
using namespace std;

int main( int argc, CHAR *argv[] )
{
   cout <<"\n ---> IO sono il processo PADRE con PID: "<<GetCurrentProcessId()<<endl;

   STARTUPINFO  si;
   PROCESS_INFORMATION  pi;

   ZeroMemory( &si, sizeof(si) );
   si.cb = sizeof(si);

   ZeroMemory( &pi, sizeof(pi) );
```

# Programma in C++ che crea un processo figlio

```
if( argc != 2 )
  {  printf("\nUsage: %s [cmdline]\n", argv[0]);
     system("pause");
     return EXIT_FAILURE;
  }
  // Start the child process.
  if( !CreateProcess( NULL,   // No module name (use command line)
     argv[1],       // Command line
     NULL,          // Process handle not inheritable
     NULL,          // Thread handle not inheritable
     FALSE,         // Set handle inheritance to FALSE
     0,             // No creation flags
     NULL,          // Use parent's environment block
     NULL,          // Use parent's starting directory
     &si,           // Pointer to STARTUPINFO structure
     &pi )         // Pointer to PROCESS_INFORMATION structure
    )
        {     printf( "\nCreateProcess failed (%d).\n", GetLastError() );
              system("pause");
              return EXIT_FAILURE;
         }
```

# Programma in C++ che crea un processo figlio

```
    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );

}
```

Listato completo del programma con messaggi per visualizzare alcuni campi significativi durante l'esecuzione del programma

## Struttura _STARTUPINFO

```
//  Specifies the window station, desktop, standard handles, and appearance
//  of the main window  for a process at creation time.
     typedef struct  _STARTUPINFO {
                DWORD  cb;
                LPTSTR lpReserved;
                LPTSTR lpDesktop;
                LPTSTR lpTitle;
                DWORD  dwX;
                DWORD  dwY;
                DWORD  dwXSize;
                DWORD  dwYSize;
                DWORD  dwXCountChars;
                DWORD  dwYCountChars;
                DWORD  dwFillAttribute;
                DWORD  dwFlags;
                WORD   wShowWindow;
                WORD   cbReserved2;
                LPBYTE lpReserved2;
                HANDLE hStdInput;
                HANDLE hStdOutput;
                HANDLE hStdError;}   STARTUPINFO, *LPSTARTUPINFO;
```

## Struttura _PROCESS_INFORMATION

// Contains **information about a newly created process** and its primary thread.
// It is used with the **CreateProcess**, **CreateProcessAsUser**, **CreateProcessWithLogonW**,
// or **CreateProcessWithTokenW** function.

```
typedef struct   _PROCESS_INFORMATION {
        HANDLE hProcess;
        HANDLE hThread;
        DWORD  dwProcessId;
        DWORD  dwThreadId;
      } PROCESS_INFORMATION,  *LPPROCESS_INFORMATION;
```