Gestione Eventi 4 WINDOWS

giovedì 27 novembre 2025 14:15

Contrariamente a UNIX in WINDOWS ci sono due categorie di eventi:

1. **Eventi di sistema**
   a. Simili ai segnali UNIX , eccetto per la modalità di processamento per tale evento
2. **Messaggi evento**
   a. Usati come strumento di notifica e comunicazione, in maniera asincrona specialmente dove i processi gestiscono reali "finestre". **Si usa il polling** (anche virtuale) : unico gestore dell'evento

Vediamo in dettaglio :

```
BOOL WINAPI HandlerRoutine( _In_  DWORD dwCtrlType )
```

Value	Meaning
CTRL_C_EVENT 0	A CTRL+C signal was received, either from keyboard input or from a signal generated by the GenerateConsoleCtrlEvent function.
CTRL_BREAK_EVENT 1	A CTRL+BREAK signal was received, either from keyboard input or from a signal generated by GenerateConsoleCtrlEvent.
CTRL_CLOSE_EVENT 2	A signal that the system sends to all processes attached to a console when the user closes the console (either by clicking Close on the console window's window menu, or by clicking the End Task button command from Task Manager).
CTRL_LOGOFF_EVENT 5	A signal that the system sends to all console processes when a user is logging off. This signal does not indicate which user is logging off, so no assumptions can be made. Note that this signal is received only by services. Interactive applications are terminated at logoff, so they are not present when the system sends this signal.
CTRL_SHUTDOWN_EVENT 6	A signal that the system sends when the system is shutting down. Interactive applications are not present by the time the system sends this signal, therefore it can be received only be services in this situation. Services also have their own notification mechanism for shutdown events. For more information, see Handler. This signal can also be generated by an application using GenerateConsoleCtrlEvent.

Attenzione alla gestione/non gestione dell'evento : dato un processo (con relativo address space) potrebbe capitare che le routine per la gestione degli errori posso essere multiple : in questo caso si usa un meccanismo di memorizzazione di queste routine basato su pile. **Quindi il kernel scorre tutta la pila del gestore degli eventi finché non c'è quello che lo gestisce in modo opportuno**. Se capita che dato un evento e nella pila di gestione degli errori non c'è nessuna routine che ritorna vero (gestendola in modo opportuno quindi) oppure vuota, si ha che viene gestito attraverso EXIT_PROCESS (shutdown di tutta applicazione). Ma come possiamo rimuovere/aggiungere gestore degli eventi?

Aggiunta o rimozione

C++

```
BOOL WINAPI SetConsoleCtrlHandler(
  _In_opt_  PHANDLER_ROUTINE HandlerRoutine,
  _In_      BOOL Add
);
```

Parameters

HandlerRoutine [in, optional]
A pointer to the application-defined **HandlerRoutine** function to be added or removed. This parameter can be **NULL**.

Add [in]
If this parameter is **TRUE**, the handler is added; if it is **FALSE**, the handler is removed.

If the *HandlerRoutine* parameter is **NULL**, a **TRUE** value causes the calling process to ignore CTRL+C input, and a **FALSE** value restores normal processing of CTRL+C input. This attribute of ignoring or processing CTRL+C is inherited by child processes.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

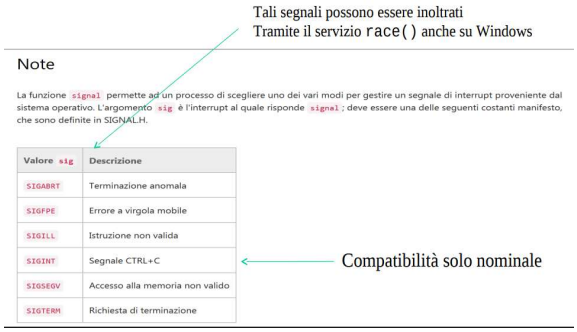
Vediamo ora come generare un evento :

Inoltro di eventi di sistema on-demand

```
BOOL WINAPI GenerateConsoleCtrlEvent( _In_  DWORD dwCtrlEvent,
                                      _In_  DWORD dwProcessGroupId );
```

Nel caso un processo sia attivato tramite `CreateProcess()` con parametro CREATE_NEW_PROCESS_GROUP
il suo identificatore corrisponde ad un identificatore di gruppo

Mentre per segnalare un evento (signal UNIX):



Vediamo un esempio :

```
// this program traps the CTRL+C system event


#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char buff[1024];

int one_shot = 1;

int  myhandler(int event){

        printf("received CTRL+C\n");
        fflush(stdout);
        while (1) {
                printf("%s\n",buff);
                if (one_shot) break;
                Sleep(1000);
        };
        return  1;
}


int _tmain(int argc, _TCHAR* argv[])
{
        BOOL result;
        result = SetConsoleCtrlHandler((PHANDLER_ROUTINE)myhandler, 1);
        printf("handler posting returned %d \n", result);
        fflush(stdout);
        while (1){
                scanf("%s", buff);
        //      break;
        }
        return 0;
}
```

Ritorno 1 nella funzione di gestione vale a dire che l'evento è gestito (non vado nello stack), evitando di chiamare EXIT_PROCESS. Se durante esecuzione genero il ctrl+c si ha ultima stringa acquisita.