# Gestione degli Errori

Marcello Barnaba <vjt@openssl.it>

# Introduzione alla gestione degli errori

- Le API di qualsiasi OS ritornano sempre un`informazione al processo che le ha chiamate, a meno di API che eseguono operazioni il cui fallimento e` impossibile
- L`informazione ritornata codificata e specifica dell`OS
- Quando si verifica un errore, la funzione stessa non puo` ritornare il codice di errore, in quanto gli errori sono tra i piu` disparati, essa ritorna un codice che indica esclusivamente che si e` verificato un errore
- Il compito di recuperare maggiori informazioni sull`errore e` affidato ad una serie di funzioni studiate per questo compito

# Il last-error code in Windows

- Il codice che le funzioni ritornano in caso di errore e` solitamente `0`, `NULL` o `-1`.
- Inoltre la maggior parte delle API settano un codice chiamato “last-error code”
- Un last-error code uguale a 0 indica successo
- Non sempre quando una funzione ha successo setta il last-error code a 0, esso e` da considerarsi affidabile solo ed esclusivamente nello statement successivo a quello in cui la funzione in esame ha ritornato errore
- Ogni thread ha un suo error code separato

# GetLastError() e SetLastError()

- **`DWORD GetLastError(VOID)`**
  La **`GetLastError()`** ritorna il valore corrente del last-error code
- **`VOID SetLastError(DWORD dwCode)`**
  La **`SetLastError()`** permette ad un thread di modificare il valore corrente del last-error code
- La **`SetLastError()`** e` intesa per essere utilizzata principalmente nelle DLL, ma anche un`applicazione puo` chiamarla (e` utile ad es. nei server di rete che fanno uso di socket in modalita` nonblocking)

# Come notificare l`utente dell`errore verificatosi

- Utilizzando la **`Beep()`** con un "beep" dallo speaker dell`elaboratore
- Utilizzando la **`MessageBeep()`** con un suono wave
- Utilizzando la **`FlashWindow()`** facendo "lampeggiare" il titolo della finestra relativa all`applicazione che si e` trovata in errore
- Utilizzando la **`MessageBox()`** per visualizzare all`utente una finestrella di messaggio

# Ottenere descrizioni dettagliate degli errori verificatisi

- Al momento della stesura di un`applicazione il programmatore si riferisce ai codici di errore predefiniti mediante delle costanti definite negli header dell`OS
- I nomi di queste costanti non sono disponibili a runtime, ma ogni codice ha una stringa associata che descrive dettagliatamente l`errore avvenuto
- Per prelevare un messaggio di errore di un generico codice di errore dalle tabelle di messaggi del sistema o di librerie ausiliarie si utilizza la **`FormatMessage()`**

# FormatMessage() (1):
## definizione

- ```
  DWORD FormatMessage(
      DWORD dwFlags,
      LPCVOID lpSource,
      DWORD dwMessageId,
      DWORD dwLanguageId,
      LPTSTR lpBuffer,
      DWORD nSize,
      va_list *Arguments
  );
  ```
- Il primo parametro e` una combinazione di flag
- Il secondo e` la sorgente della definizione messaggio
  ```

# `FormatMessage()` (2): parametri

- Il terzo parametro e` il codice del messaggio da leggere
- Il quarto parametro e`la lingua in cui si deve prendere il messaggio
- Il quinto parametro e` il buffer che accogliera` il messaggio letto
- Il sesto parametro e` la dimensione del buffer
- Il settimo parametro e` un puntatore ad una lista di valori che verranno inseriti a runtime nel messaggio

# FormatMessage() (3):
## flag

- **`FORMAT_MESSAGE_ALLOCATE_BUFFER`:**
  Indica alla funzione di allocare un buffer di dimensione **`nSize`** per accogliere la stringa. Il processo chiamante dovrebbe liberare questo buffer usando la **`LocalFree()`**
- **`FORMAT_MESSAGE_FROM_STRING`:**
  Indica alla funzione che il parametro **`lpSource`** e` la stringa sorgente del nostro messaggio di errore
- **`FORMAT_MESSAGE_FROM_SYSTEM`:**
  Indica alla funzione che il messaggio va cercato nelle tabelle di stringhe del sistema operativo stesso
- **`FORMAT_MESSAGE_FROM_HMODULE`:**
  Indica alla funzione che **`lpSource`** e` un handle ad un modulo contenente una tabella di stringhe in cui cercare il codice specificato

# Esempio di codici di errore in Windows

| | | |
|---|---|---|
| 0 | The operation completed successfully | ERROR_SUCCESS |
| 1 | Incorrect function | ERROR_INVALID_FUNCTION |
| 2 | The system cannot find the file specified | ERROR_FILE_NOT_FOUND |
| 3 | The system cannot find the path specified. | ERROR_PATH_NOT_FOUND |

- Il primo campo specifica il codice ritornato dalla `GetLastError()`, il secondo la stringa ad esso associata, ed il terzo la costante simbolica
- Gli errori di sistema hanno sempre il 29mo bit uguale a 0

# `FatalAppExit()`

- L`ultima risorsa per terminare un`applicazione in un grave stato di errore
- Dovrebbe essere utilizzata solo quando non ci sono altri modi per terminare
- **`FatalAppExit()`** potrebbe non liberare completamente le aree di memoria allocate all`applicazione, non chiudere i file aperti e generare instabilita` nel sistema
- Un`applicazione in grave stato di errore dovrebbe liberare tutta la memoria allocata, chiudere tutti i propri file e ritornare dalla **`WinMain()`**