Piattaforme di esecuzione: cenni introduttivi

Programmazione di Sistema

A.A. 2016-17

Argomenti

- Ambienti operativi
  - Windows
  - Linux
  - Android
- Interfacciarsi con il sistema operativo
- Gestione degli errori

Linguaggi e ambienti di sviluppo

- Linguaggio C e C++, con estensioni dello standard 2O11 (C++Ox)
  - VisualStudio 2O12+ (Windows)
  - g++ 4.8.x+ (Linux)
- Linguaggio Java
  - AndroidStudio

Interfacciarsi con il sistema operativo

- Fatta eccezione per i sistemi più elementari, l'esecuzione di un'applicazione avviene nel contesto di un S.O.
  - Che offre un insieme di servizi, funzionalità, convenzioni che ne permettono il funzionamento
- Per sfruttare tali servizi, un'applicazione deve conformarsi alle specifiche del S.O.
  - Sia a livello di codice sorgente
  - Che a livello di codice eseguibile

API – Application Programming Interface

- Definisce un insieme di funzioni e di strutture dati che vengono offerte, così come sono, al programmatore
  - Invocandole, ottiene l'accesso ai servizi offerti dal sistema sottostante
- Le API effettivamente offerte dai sistemi operativi (System Call) sono prevalentemente annegate in funzioni di libreria
  - Che ne astraggono l'utilizzo
  - Es.: C Standard Library

ABI – Application Binary Interface

- Definisce quale formato debba avere un prodotto software per essere compatibile con il S.O.
  - E a quali convenzioni debba sottostare
- Comprende aspetti quali
  - Convenzioni di chiamata e passaggio dei parametri
  - Uso dei registri del processore
  - Innalzamento di privilegio e invocazione del S.O.
  - Collegamento tra moduli e struttura dei file binari

ABI – Application Binary Interface

- È supportata dagli strumenti che compongono la toolchain
  - Compilatore
  - Linker
  - Debugger
  - Profiler
  - Inspector
- Può essere oggetto diretto della programmazione di sistema
  - Quando si utilizzano meccanismi quali il caricamento dinamico di moduli, l'emulazione delle piattaforme di esecuzione, …

Interfacciarsi con il S.O.

- Per sviluppare un programma che si interfacci direttamente con il sistema operativo, occorre
  - Conoscere le API da chiamare
  - Le strutture dati coinvolte
  - Le convenzione generali definite dal S.O.

Accedere alle API

- Occorre dichiarare le funzioni e definire i tipi dei parametri
  - Attraverso l'inclusione di appositi file header
  - Nel caso di Windows, di solito equivale ad includere il file Windows.h
- In alcuni casi, può essere necessario collegare l'eseguibile con librerie specifiche
  - Ad esempio, in Linux, un programma concorrente deve essere collegato alla libreria PThread

Convenzioni

- Ogni S.O. mantiene, al proprio interno, un insieme di strutture dati che descrivono lo stato del sistema
  - Tali strutture NON sono esposte direttamente al programmatore
- Le API permettono di accedere in modo indiretto a tali strutture attraverso riferimenti opachi
  - Detti HANDLE in Windows
  - FileDescriptor in Linux

Gestione degli errori

- Le funzioni invocate possono avere successo o fallire
  - Occorre verificarne sempre l'esito
  - Il modo di farlo dipende dal S.O.

Gestire gli errori in Windows

- Per sapere se la chiamata di una API ha avuto successo o meno, occorre basarsi sul tipo di ritorno
  - Adottando una strategia opportuna
- BOOL (int)
  - Restituisce 0 in caso di fallimento, qualunque altro valore in caso di successo
- HANDLE (void*)
  - Restituisce 0 o -1 in caso di fallimento: consultare la documentazione online
- PVOID (void*)
  - NULL indica un fallimento
  - Un puntatore valido indica successo
- LONG/DWORD (unsigned long)
  - Dipende dal significato del valore ritornato: consultare la documentazione online

Gestire gli errori in Windows

- Appurato che si è verificato un errore, occorre capire quale sia
  - Si invoca la funzione DWORD GetLastError();
  - Poiché ulteriori chiamate potrebbero nascondere l'errore precedente, occorre invocarla il prima possibile
- Il valore ritornato è un numero a 32 bit
  - Per conoscerne il significato è possibile utilizzare
    la utility ErrorLookup di VisualStudio
  - O invocare la funzione FormatMessage(…)

Gestire gli errori in Linux

- I dettagli con cui una API indica il fallimento della richiesta sono variabili
  - Prevalentemente, un valore pari a -1 indica il fallimento
  - Occorre controllare la documentazione online
- Si accede al codice di errore ispezionando il contenuto della pseudo-variabile globale errno
  - #define errno (*__errno_location ())
- Come nel caso di Windows, occorre ispezionare il contenuto di tale variabile non appena si verifica l'errore
  - Ulteriori chiamate al sistema potrebbero sovrascriverla

Gestire gli errori in Linux

```
if (fsync (fd) == -1) {

  // fprintf chiama altre system call,
  // che sovrascrivono errno
  fprintf (stderr, "fsync failed!\n");

  if (errno == EIO) // NON VA BENE!!!
    fprintf (stderr,
      "I/O error on %d!\n", fd);
}
```


Uso di stringhe

- La codifica e gestione del testo è un argomento molto più complesso di quanto appaia a prima vista
  - La rappresentazione dei caratteri a 8 bit (char) è insufficiente per la maggior parte degli alfabeti

Codifica dei caratteri

- Il consorzio Unicode ha definito una rappresentazione standard che prevede alcuni milioni di simboli rappresentati da numeri interi
  - La loro codifica richiede almeno 21 bit
- Rappresentazioni possibili
  - UTF-32: ogni simbolo occupa 32 bit
  - UTF-16: un simbolo può occupare una o due parole da 16 bit
  - UTF-8: un simbolo può occupare da una a quattro parole di 8 bit

Codifica dei caratteri

- Le rappresentazioni a lunghezza variabile possono avere ordinamenti differenti
  - Big endian, little endian
- I primi 128 valori Unicode coincidono con la codifica ASCII
- Nei primi 65000 valori sono codificati la maggior parte dei sistemi di scrittura correnti
  - Per evitare la complessità di gestire stringhe a lunghezza variabile, spesso si adotta una codifica a 16 bit, ignorando caratteri rari

Caratteri in Windows

- Due tipi base
  - char: 8bit, codifica ASCII estesa
  - wchar_t: 16 bit, codifa UNICODE
    - Basic Multilingual Plan 0
- Tipo generico
  - TCHAR: definito nel file tchar.h
- Tipi derivati
  - LPSTR: sequenza terminata da \0 di char
  - LPWSTR: sequenza di wchar_t
  - LPTSTR: sequenza di TCHAR
  - Le versioni con «LPC-» indicano sequenze immutabili

Caratteri in Windows

- Tutte le API del sistema che prevedono l'uso di caratteri e stringhe sono offerte in due versioni
  - ASCII, usa suffisso "-A"
  - WCHAR, usa suffisso "-W"
- Si invoca solo la versione generica, senza suffisso

Esempio

```
BOOL CreateDirectoryA(
        LPCSTR lpPathName,
        LPSECURITY_ATTRIBUTES lpSecAtt);
BOOL CreateDirectoryW(
        LPCWSTR lpPathName,
        LPSECURITY_ATTRIBUTES lpSecAtt);
#ifdef UNICODE
#define CreateDirectory CreateDirectoryW
#else
#define CreateDirectory CreateDirectoryA
#endif // !UNICODE
```


Caratteri in Linux

- Per default, il compilatore GCC codifica eventuali caratteri non-ASCII con la codifica UTF-8
  - Poiché la stringa che ne deriva termina correttamente con \0, questo non crea problemi al kernel
- La codifica UTF-8 può creare problemi in caso di algoritmi semplicistici
  - La funzione *strlen(str)*, ad esempio, non indica più il numero di caratteri effettivamente presenti, ma solo il numero di byte non nulli
  - Si determina il numero di caratteri con la funzione *mbstowcs(NULL, str, 0)* *(MultiByteStringTOWideCharacterString)*

Caratteri in Linux

- Il tipo wchar_t esiste, ma ha lunghezza pari a 4 byte
  - Può ospitare qualsiasi carattere Unicode
  - Sequenze costanti di wchar_t sono precedute da «L »

```
wchar_t* s= L"Ω€® ™åßðƒ∞Δªøπº¬∑∫µ"
```