HOWTO zu Unterschieden bei Hardware- und Software-Systemen

(C) 2007-2017 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: hwsw-differences-HOWTO.txt,v 1.19 2019/11/26 19:37:07 tsbirn Exp $

Dieses Dokument beschreibt die Hardware- und Software-Unterschiede diverser Rechnersystem und Programmiersprachen bzw. Compiler von Programmiersprachen. Teilweise handelt es sich dabei um feste nicht umgehbare Vorgaben, teilweise um Konventionen/Regeln, die nur in begründeten und dokumentierten Ausnahmefällen durchbrochen werden sollten.

INHALTSVERZEICHNIS

1) Einführung
2) Inhalt von Text-Dateien (Quellcode)
3) Zugriff auf Dateien
4) Zahlen- und Zeichenketten
5) Variablen, Zeiger (Adressen) und Arrays
6) Byte-Sex (Little Endian <-> Big Endian)
7) Datenbanken

---

1) Einführung

Zwischen Rechner-Hardware (PC, UNIX-Workstation, HOST) und Programmiersprachen bzw. Compilern von Programmiersprachen (C, Perl, COBOL, Assembler) gibt es viele Unterschiede, die beim Transfer von Programm-Quellcode und von Daten (z.B. über Netzwerk!) zwischen verschiedenen Rechnern und beim Kombinieren verschiedener Programmiersprachen zu einem Programm auf dem gleichen Rechner zu berücksichtigen sind.

Diese Punkte sind NUR DANN während eines Programmlaufs kritisch, wenn Daten zwischen unterschiedlichen Programmiersprachen bzw. Compilern von Programmiersprachen ausgetauscht werden, die z.B. zu einem Executable oder einer Phase zusammengelinkt/gebunden wurden oder Daten zwischen mehreren Programmen übergeben werden, die in unterschiedlichen Programmiersprachen realisiert sind.

Beim Austausch von Daten über eine Datenbank bestehen obige Probleme normalerweise nicht. Eine Datenbank legt die Daten sowieso in einem eigenen Format ab. Beim Lesen/Schreiben von Daten aus/in eine Datenbank wird durch das jeweilige API (Application Programmers Interface) automatisch zwischen dem datenbankspezifischen Format und dem Format der konkreten HW-Plattform bzw. dem der konkreten Programmiersprache konvertiert.

2) Inhalt von Text-Dateien (Quellcode)

* Zeichencodierung (gilt bereits für Programmcode)
  + Dos/Windows: Latin1 (8 Bit) bzw. UCS-2 (Unicode, 2 Byte)
    Terminal (CMD.exe) und GUI haben unterschiedlichen Zeichensatz
  + Linux/Unix: Latin1 (8 Bit) bzw. UTF-8 (Unicode, 1-4 Byte)
  + Apple Mac: UTF-8 (eigene MAC-Variante!, 1-4 Byte)
  + HOST: EBCDIC (7 Bit -> 8 Bit geändert)

* Zeilenenden in Textdateien (gilt bereits für Programmcode)
  + Unix/Linux: Zeilenende = "\n" (nur Newline=Linefeed=10, C-Zeilenmodell)
  + Dos/Windows: Zeilenende = "\r\n" (Carriage Retuern=13 + Newline=10)
  + Apple Mac: Zeilenende = "\r" (nur Carriage Return=13)
  + HOST: a) Feste Zeilenlänge, mit Leerzeichen aufgefüllt
    b) 4 Byte Satzlänge vor jedem Satz (2x 0-Byte + 2 Byte Länge)

* Dateiende in Textdateien und zum Abschluss einer interaktiven Eingabe vom Terminal
  + Dos/Windows: Strg-Z (veraltet, aber kommt noch in Dateien vor)
  + Unix/Linux: Strg-D (nur Terminal-Eingabe-Abschluss, nicht in Dateien)
  + Apple Mac: Strg-D
  + HOST:

3) Zugriff auf Dateien

```
----------------------
* Dateinamen und Dateipfade
  + Dos/Windows: C:\PFAD\...\ZU\DATEI.txt
                 Laufwerksbuchstabe A: .. Z: am Anfang
                 9 verbotene Zeichen: ? " / \ > < * | :   (Null-Byte
                 GROSS/kleinschreibung beim ANLEGEN/UMBENENNEN nicht egal
                 GROSS/kleinschreibung beim ZUGRIFF egal
  + Unix/Linux:  /PFAD/.../ZU/DATEI.txt
                 Keine Laufwerks-Buchstaben (von Plattenphysik wird abstrahiert)
                 2 verbotene Zeichen: / (Verz.trenner) \0 (Null-Byte)
                 GROSS/kleinschreibung relevant
  + Apple Mac:   /PFAD/.../ZU/DATEI.txt
                 Keine Laufwerks-Buchstaben (von Plattenphysik wird abstrahiert)
                 2 verbotene Zeichen: / (Verz.trenner) \0 (Null-Byte)
                 GROSS/kleinschreibung beim ANLEGEN/UMBENENNEN nicht egal
                 GROSS/kleinschreibung beim ZUGRIFF egal
  + HOST:        :PUBSET:$KENNUNG.DATEINAME
                 Erlaubte Zeichen: A-Z 0-9 - .
                 GROSS/kleinschreibung egal
                 Verbotene Zeichen: Win-Z. _ #
                 Für temporäre Dateien und Kennungen: # @ $

* Dateilocking (Sperren gegen konkurrierenden Zugriff)
  (kein Locking -> der letzte hat Recht)
  + Dos/Windows: JA:   Automatisch (mandatory=erzwungen) durch Betriebssystem
                 JA:   Manuell (advisory=empfohlen) zusätzlich möglich
  + Unix/Linux:  NEIN: Automatisch (mandatory=erzwungen) durch Betriebssystem
                 JA:   Manuell (advisory=empfohlen) durch Anwendung (zu programmieren)
  + Apple Mac:
  + HOST:        Automatisch (mandatory=erzwungen) durch Betriebssystem


4) Zahlen- und Zeichenketten
----------------------------
* Zahlendarstellung als Binärzahlen <-> BCD-Zahlen (Binary Coded Digits)
  + C:     Binäre Codierung (Vorzeichen + Mantisse + Exponent)
           Wissenschaftliche Anwendungen, möglichst viele Stellen,
           Im 10er-System exakt darstellbare Zahlen evtl. nicht exakt darstellb.
           (nur Summe von 2-er Potenzen, d.h. 0.125 0.25 0.5 1 2 3 ...)
  + COBOL: BCD-Codierung (Vorzeichen + Vorkommateil + Komma + Nachkommateil)
           Kaufmännische Anwendungen, keine Darstellungsfehler im 10er-System
           Gepackt:   1 Zeichen pro Nibble (4 Bit) = 2 Zeichen pro Byte (8 Bit)
           Ungepackt: 1 Zeichen pro pro Byte (8 Bit)
           Vorzeichen "+-" extra in 1 Nibble oder 1 Byte dargestellt

* Art der binären Ganzzahlendarstellung
  + Einer-Komplement:              Binär: -N === N mit allen Bits gekippt
                                   (symmetrisch, zwei Nullwerte +0 und -0)
  + Zweier-Komplement:             Binär: -N === N mit allen Bits gekippt + 1
                                   (asymmetrisch, ein Nullwert)
                                   Inzwischen fast ausschließlich verwendet
  + Vorzeichen-Bit + Absolutwert: Nicht gebräuchlich

* Zahlendarstellung Festkomma <-> Fließkomma
  + C:         Position des Dezimalkommas variabel (Anzahl Stellen gesamt fix)
               Codierung im Binärsystem (Zahlen außer 2er-Potenz-Summen 1/2 1/4
               1/8 ... prinzipiell nicht exakt darstellbar)
               Wissenschaftliche Rundung bei Ausgabe (< 0.5 -> 0, >= 0.5 -> 1)
               ("wissenschaftliche Art", Komma "wandert", IEEE-754 Standard)
  + COBOL/ASS: Anzahl Nachkommstellen fix ("kaufmännische Art"),
               Anzahl Vor- und Nachkommastellen bei Variablendefinition festleg.
               Darstellung der Zahlen im Dezimalsystem (Zahlen außer 2er+5er-
               Potenz-Summen prinzipiell nicht exakt darstellbar)
               Kaufmännische Rundung beim Rechnen und Ausgabe (BCD-Darstellung)

* "Breite" von Zahlen (möglicher Zahlenbereich)
  + C:         Fixer Set von Ganzzahl- (char, short, int, long, long long)
               und Fließkomma-Datentypen (float, double, long double)
               Breite von int/long von Prozessor-Wortbreite abhängig
               float/double nach IEEE-754 Norm = identisch auf allen Plattformen
  + COBOL/ASS: Definierbare Breite von Zahlen (Anzahl Vor+Nachkommastellen)
               da als BCD-Zahlen dargestellt (Anweisung PIC)
               Gepackt = 2 Ziffern pro Byte (platzsparend aber langsam)
               Ungepackt = 1 Ziffer pro Byte (schnell aber platzsverschwendend)

* Darstellung von Strings (Zeichenketten) im Speicher
```

  + C: String variabler Länge, danach als Begrenzung ein Null-Byte \0
               (Länge implizit, Speicherbedarf 1 größer, KEIN Null-Byte mögl.)
  + Perl: 4 Byte Länge (wo?) + String variabler Länge (0-Byte erlaubt)
  + COBOL/ASS: Fixe Länge + mit Leerzeichen aufgefüllt
               (Länge woanders hinterlegt, Speicherbedarf immer gleich)
  + HOST: 2/4 Byte Länge am Anfang + danach String variable Länge
               (Länge explizit, Speicherbedarf um 2/4 Byte größer)
  + PASCAL: 1/2/4 Byte Länge am Anfang + danach String variable Länge
               (Länge explizit, Speicherbedarf um 1/2/4 Byte größer)

5) Variablen, Zeiger (Adressen) und Arrays

* Reihenfolge der Funktionsparameter bei einem Funktionsaufruf
  + C: Von "links nach rechts" (da variabel lange Listen möglich)
  + COBOL/ASS: Von "rechts nach links" (effizienter)

* Wer räumt Stack am Ende eines Funktionsaufrufes auf (Parameterübergabe)
  + C: Aufgerufene Funktion
  + COBOL/ASS: Aufrufer

* Globale <-> Lokale Variablen
  + C: Globale (Text-Segment) + Lokale Variablen (Stack-Segment)
               Beschränkung auf Programm, Datei(Modul), Funktion/Block möglich
               Statische (Text-Segment) + Dynamische Variable (Heap)
               Schutz gegen versehentlichen Zugriff
               Kein Schutz gegen versehentlichen Zugriff sobald Zeiger verwendet
  + COBOL/ASS: Parameterleiste = Globale Variablen
               Kein Schutz gegen versehentlichen Zugriff
               Lokale Variablen per "USING" nur in ASSEMBLER
               COBOL: Linkage Section: Speicherbereich "durchreichen" oder nicht

* Zeiger (Pointer) bzw. Adressen
  + C: "Typisiert": "wissen" worauf sie zeigen
               Compiler überprüft typkompatible Verwendung (außer bei Casts)
               Compiler überprüft nicht Zugriffe per Zeiger auf Grenzen
  + COBOL/ASS: "Untypisiert: "wissen nicht" worauf sie zeigen (reine Adresse)

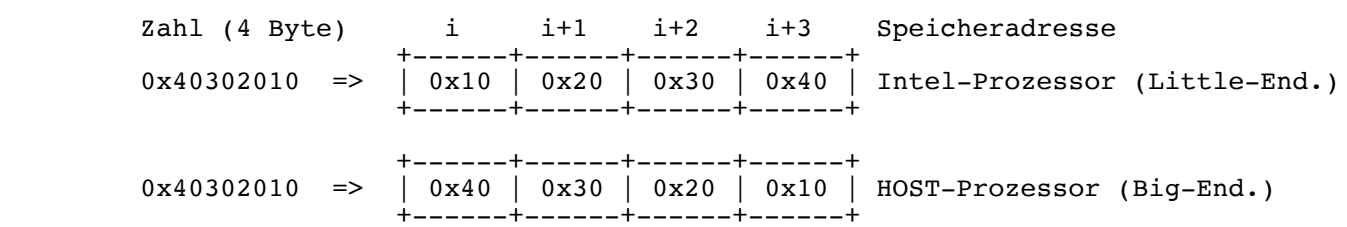
* Array-Länge
  + C: statisch (außer durch Emulation per Zeiger + Heap-Speicher)
  + ASS: statisch
  + COBOL: statisch

* Array-Indices
  + C: 0 bis Arraylänge - 1
  + ASS: 0 bis Arraylänge - 1
  + COBOL: 1 bis Arraylänge

* Dimensionalität von Arrays
  + C: Beliebig (1-n dimensional)
  + ASS: 1-dimensional (mehrdimensional manuell emulieren)
  + COBOL: 1-dimensional

6) Byte-Sex (Little Endian <-> Big Endian)

* Anordnung zusammengehörender Bytes im Speicher bei Datentypen mit mehr als
  1 Byte Länge
  + i386: Niedrigwertigstes Byte an Speicherstelle mit niedrigster Adresse
          (least significant byte first)
  + HOST: Höchstwertigstes Byte an Speicherstelle mit niedrigster Adresse
          (most significant Byte first)



Dies betrifft z.B. auch IP-Adressen: Die IP-Adresse 192.168.1.2 wird im 1. Fall
als (2, 1, 168, 192) = (0x02, 0x01, 0xA8, 0xC0) und im 2. Fall als (192, 168,
1, 2) = (0xC0, 0xA8, 0x01, 0x02) in vier aufeinanderfolgenden Bytes des

Speichers abgelegt.

Es gibt sogar HW-Plattformen (z.B. MIPS/PowerPC), die sich zwischen Big und Little-Endian umschalten lassen (teilweise zur Laufzeit, z.B. Context-Switch).

Beide Probleme sind nicht während EINES Programmlaufs kritisch, sondern nur dann, wenn solche Daten auf Platte gespeichert und wieder gelesen werden oder per Netzwerk ausgetauscht werden. Dann müssen die Daten in die eine oder andere Richtung konvertiert werden (z.B. mit Hilfe des altbekannten Verfahrens XDR = eXternal Data Representation von SUN, das in Form von Bibliotheken verfügbar ist) oder man konvertiert manuell durch Byte-Tausch.

Allerdings kann man die notwendigen Tauschaktionen nicht am "Strom der binären Daten" selbst ablesen, da diese nur eine Folge von Bytes ohne Typinformation bilden. Sondern man muss "wissen", was die gerade transferierten Bytes bedeuten. D.h. welche Anzahl von Bytes ein "Objekt" (String, Integer, Float, Struktur, ...) darstellt und welchen Datentyp dieses Objekt repräsentiert. D.h. man muss eine Strukturbeschreibung der "Objekte" auf beiden Plattformen haben.

TCP/IP Netzwerk-Parameter in numerischer Form (IP-Adressen, Ports, ...) werden immer im Big-Endian-Format dargestellt. Die C-Bibliothek bietet Funktionen namens htonl (host to network long), htons (host to network short), ntohl (network to host long) und ntohs (network to host short) an, um zwischen der Byte-Reihenfolge des Systems und des Netzwerks zu konvertieren. Bei Big-Endian HW-Plattformen sind diese Funktionen ohne Wirkung, da die Byte-Reihenfolge bereits im Netzwerk verwendeten entspricht.

7) Datenbanken

* Verschiedene Systeme verwenden unterschiedliche Art von Datenbank-Zugriffen
  + C: SQL-Datenbank
    Unsortierte Mengen von Tupeln, per Schlüssel adressierbar
    Präcompiler für ESQL (Embedded SQL)
    Dynamisches SQL (String zur Laufzeit zusammenbauen) auch möglich
  + COBOL/ASS: ISAM-Dateien (Index Sequential Access Method)
    Records in bestimmter Reihenfolge, per Nummer adressierbar
    Umgestellt auf SQL-Datenbank
    Präcompiler für ESQL (Embedded SQL)