Kodowanie znaków narodowych

Zbigniew Jurkiewicz
Instytut Informatyki MIMUW

25 maja 2015

1 Podejścia do uniwersalizacji oprogramowania

Są trzy poziomy „umiędzynarodowienia”:

- lokalizacja (*localization*) — L10N
- internacjonalizacja (*internationalization*) — I18N
- wielojęzykowość (*multilingualization*) — M17N

Lokalizacja to adaptacja systemu do lokalnej kultury i języka użytkownika, obejmuje ona powszechnie znane cechy, jak

- kodowanie znaków narodowych
- czcionki ekranowe i drukarkowe
- sposób wprowadzania danych z klawiatury

Inne ważne cechy to:

- separator dziesiętny dla liczb
- reprezentacja dat i czasu, waluty
- dzielenie wyrazów przy przenoszeniu
- kolejność sortowania

Lokalizacja zwykle jest robiona *ad hoc*, przez doraźne modyfikowanie kodu programów. Lepiej jednak dokonać takiej adaptacji systemu, aby wydzielić elementy zależne od lokalizacji w osobne „bazy danych” Lokalizacja polega wtedy na zmianie ich zawartości, bez dotykania programów. Mówimy wtedy o internacjonalizacji.

Terminu wielojęzykowość używa się dla przystosowania oprogramowania do *równoczesnego* użycia w *wielu* językach. Jest to najtrudniejsze, bo wymaga dopasowania do kontekstu.

2 Przetwarzanie tekstów

Przy przetwarzaniu tekstów mamy do czynienia z trzema poziomami abstrakcji

- zestaw znaków (*character set*)
- sposób kodowania (*encoding*)
- czcionki (*fonts*)

2.1 Historia

Pierwsze używane kody znaków były 7-bitowe (podobne do ASCII). Ósmy bit często wykorzystywano do kontroli poprawności itp. Powodowało to (i czasem wciąż powoduje) kłopoty przy przesyłaniu siecią.

Obecnie większość oprogramowania jest *8-bit clean* — poprawnie przetwarza 8-bitowe kody znaków. Powstał jednak nowy problem: *endianness* — wybór kolejności bajtów dla kodów wielobajtowych (np. Unicode).

2.2 Znaki i zestawy znaków

Znak (*character*) to pojęcie abstrakcyjne. Znaki są łączone w *zestawy znaków* (*character sets*), z ewentualnym przypisaniem numeru (indeksu) znaku w ramach zestawu.

Uwaga: termin *alfabet* jest w tym kontekście rzadko używany.

Proces kodowania znaków przebiega dwuetapowo:

- Odwzorowanie ze znaków zestawu na liczby całkowite zwane punktami kodowymi (*Code Points*, CP), każdemu znakowi odpowiada pojedynczy numer.
- Odwzorowanie z kodów CP na *oktety*, określające sposób ich reprezentacji. Dla języków „zachodnich" wystarcza pojedynczy oktet. Dla języków „wschodnich" (chiński, japoński) potrzeba więcej.

Dla zestawów znaków wymagających więcej niż jednego oktetu stosuje się dwa typy kodowania:

- jedna ustalona liczba oktetów dla wszystkich znaków (*wide character format*)
- ciągi oktetów o zmiennej długości (*multibyte character format*)

Kody o zmiennej długości dają oszczędność pamięci, ale spowalniają niektóre operacje.

Przykładami kodów o stałej szerokości są oczywiście wszystkie kody jednooktetowe, np. ASCII, ISO-8859-2. Inne przykłady

- Kod Unicode (ISO-10646/UCS-2) — 2 oktety
- JIS dla języka japońskiego — 2 oktety
- Kod ISO-10646/UCS-4 — 4 oktety

Dla kodów o zmiennej długości dąży się do zachowania kodów ASCII jako pojedynczych oktety. Daje to zgodność z obecnymi (jednooktetowymi) programami. Najpopularniejszy przykład takich kodów to kody transformacyjne Unicode, np. UTF-8, służące do przekodowania Unicode na postać ciągu oktetów.

Inny przykład to Unixowy kod EUC-JP dla języka japońskiego. W nim także zachowane są kody dla ASCII, zaś pozostałe kody reprezentuje się dwoma oktetami, z ustawionym najwyższym bitem każdego oktetu.

2.3 Fonty

Kształt (obraz) służący do wyświetlania lub drukowania znaku będziemy nazywać (za J.S. Bieniem) *glifem* (*glyph*). Termin „litera" (itp.) jest nieodpowiedni, ponieważ w niektórych językach (np. arabskim) dla pojedynczego znaku są używane różne glify zależnie od położenia znaku (na początku, końcu czy w środku słowa) Ponadto jeden glif może odpowiadać ciągowi znaków (tzw. ligatury), np. „fi".

Glify łączy się w *fonty* — zbiory glifów o wspólnych cechach graficznych. Fonty to odpowiednik *czcionek* z europejskiej typografii kontynentalnej, gdzie rozróżnia się pojęcia:

- *kroju pisma*, np. Bodoni Poster
- konkretnej *czcionki*, np. Bodoni Poster 14pt

2.4 Zestaw i kod ASCII

Oryginalna nazwa to *American Standard Code for Information Interchange*, jest to kod 7-bitowy. Kody „normalnych" (drukowalnych) znaków są zawarte w zakresie 32–126. Oprócz tego istnieją kody *sterujące* w zakresie 0–31, m.in. znaki nowej linii, tabulacji.

Istnieją różne odmiany kodu ASCII, standaryzowane jako ISO 646, np. US-ASCII (tzw.międzynarodowa wersja referencyjna) ma oznaczenie ISO 646-IRV.

2.5 Rodzina zestawów ISO 8859

Jest to rozszerzenie ASCII na kody 8-bitowe (pełny oktet), popularnie (choć niepoprawnie) nazywane Latin-*n*. Znaki są kodowane w jednym oktecie, maks. 96 dodatkowych znaków o kodach 160–255. Kodowanie jest zgodne z normą ISO 2022.

Podstawowa wada to arbitralnie (i niewygodnie) podzielone znaki wg kryteriów geograficznych (syndrom „żelaznej kurtyny" [J.S. Bień]).

Standard	Nazwa	Języki
ISO 8859-1	Latin-1	zachodnia Europa
ISO 8859-2	Latin-2	wschodnia Europa
ISO 8859-3	Latin-3	bałkańskie i inne
ISO 8859-4	Latin-4	skandynawskie i bałtyckie
ISO 8859-5	Cyrillic	rosyjski, ukraiński
ISO 8859-6	Arabic	arabski
ISO 8859-7	Greek	grecki
ISO 8859-8	Hebrew	hebrajski
ISO 8859-9	Latin-5	turecki
ISO 8859-10	Latin-6	lapońskie, nordyckie i eskimoskie

2.6 Unicode i ISO 10646

Są to uniwersalne zestawy znaków (UCS, *Universal Character Sets*), zgodne z ISO 8859-1 (Latin-1). Unicode jest 2-oktetowym standardem przemysłowym. ISO 10646 ma dwa warianty

- UCS-4, 4-oktetowy
- UCS-2, 2-oktetowy, podzbiór poprzedniego

UCS-2 i Unicode 2.0 zostały zunifikowane. Około 39000 pozycji zostało wykorzystanych, 18000 pozostaje wolnych. Niestety występują problemy dla języków azjatyckich. Unicode zawiera „zunifikowany Han", ale to tylko 21000 znaków. W dodatku kodowanie utożsamia znaki o całkowicie odmiennym znaczeniu w różnych językach, jedynie ze względu na wspólną postać graficzną.

Kodowanie UCS-4 obejmuje 4 oktety (31 bitów), co daje 2 miliardy znaków. Zbiór pozycji kodowych jest podzielony na *płaszczyzny* (*plane*). Pierwsza płaszczyzna (*Basic Multilingual Plane*) to Unicode/UCS-2 i tylko ona jest dobrze określona.

2.7 Zestawy transformacyjne

UTF — UCS/Unicode Transformation Format

- Dodatkowe przekodowania UCS-2/UCS-4 na postać „strawną" dla obecnych programów
- UTF-16 — prawie zgodny z UCS-2
- UTF-8 — obecnie bardzo popularny, m.in. w przeglądarkach

Zestaw UTF-8

- Kodowanie zmiennej długości, 1–6 oktetów (praktycznie do 4), zgodne z ASCII
- Oktet zerowy nie używany.
- Kody ASCII nigdy nie występują gdy więcej niż jeden oktet. Górny bit zero.
- Pierwszy oktet wskazuje liczbę oktetów (liczbą ustawionych górnych bitów).
- Dalsze oktety mają dwa górne bity 10.

2.8 Prywatne zestawy znaków

Różne zestawy narodowe lub stosowane przez poszczególnych producentów. Większość rozszerza ASCII, ale niektóre niezgodne ani z ASCII, ani ISO 2022.

Przykłady: KOI-8 dla rosyjskiego, różne zestawy Microsoftu, EBCDIC IBM.

Rozszerzanie zestawu znaków w ISO 2022

- Pochodzi od ECMA (European Computer Manufacturer's Organization).
- Unicode nie zawsze wystarcza, UCS-4 marnuje miejsce.
- Dopuszczenie kilku zestawów znaków w jednym strumieniu danych.
- Zasada: podział kodu 8-bitowego na górną i dolną połówkę, niezależne wybieranie zestawów dla połówek przy pomocy sekwencji sterujących.

3 Linux

Standard SUS3 (dawniej POSIX) obejmuje model *locale*, gdzie wyboru lokalizacji dokonuje się przez ustawienie zmiennych środowiska, np.

```
LANG=pl_PL.iso8859-2
```

przy czym

- `pl` to wybór języka (skrót wg normy ISO 639)
- `PL` to wybór kraju (wg ISO 3166)
- `iso8859-2` to używany zestaw znaków

W Unixie zestawy dopasowań do lokalnych zwyczajów noszą nazwę *locale* i obejmują takie rzeczy jak zestaw znaków i ich klasyfikacja (np. dla funkcji `isalpha()`), reguły sortowania (*collating rules*) czy sposób zapisu daty i czasy. Wybiera się je zmiennymi środowiska (np. `LC_COLLATE`, `LC_CTYPE`, `LC_ALL` czy `LANG`), zaś opisane są one w podkatalogach `/usr/share/locale` lub tp.

Wartości tych zmiennych środowiska mają postać

język_terytorium.kodowanie@modyfikator

na przykład

de_DE.utf-8@euro

lub

pl_PL.utf-8

Poza językiem reszta jest opcjonalna.

Polecenie shella `locale` podaje aktualny lokal

```
[zbyszek@katastrofa ~]$ locale
LANG=pl_PL.UTF-8
LC_CTYPE="pl_PL.UTF-8"
LC_NUMERIC="pl_PL.UTF-8"
LC_TIME="pl_PL.UTF-8"
LC_COLLATE="pl_PL.UTF-8"
LC_MONETARY="pl_PL.UTF-8"
LC_MESSAGES="pl_PL.UTF-8"
LC_PAPER="pl_PL.UTF-8"
LC_NAME="pl_PL.UTF-8"
LC_ADDRESS="pl_PL.UTF-8"
LC_TELEPHONE="pl_PL.UTF-8"
LC_MEASUREMENT="pl_PL.UTF-8"
LC_IDENTIFICATION="pl_PL.UTF-8"
LC_ALL=
[zbyszek@katastrofa ~]$
```

Aby program w C korzystał z tych (lub innych) ustawień, należy wywołać funkcję `setlocale()`:

```
#include <locale.h>
...
  if (setlocale(LC_ALL, "") == NULL)
    error(...);
```

Zwraca ona statyczny wskaźnik do napisu identyfikującego lokal, dla błędów zwraca NULL.

Pierwszy argument to *kategoria* — wartości jak nazwy zmiennych środowiska. LC_ALL ma najwyższy priorytet, LANG najniższy.

Drugi argument podaje nową wartość:

- `NULL`: nie ustawiać, używany gdy chcemy się dowiedzieć o bieżącą wartość;
- `""`: pobrać ze środowiska;
- inny: ustawić, powinien to być podkatalog z lokalem.

Edytor tekstów GNU Emacs, od wersji 20 zawiera na stałe pakiet MULE (*Multi-Lingual Extension Emacs*)

W bibliotece języka C istnieje funkcja `gettext()` do łatwego przerabiania komunikatów w istniejących programach na wersje wielojęzykowe.

4 Literatura

- Janusz S. Bień *Język polski w sieciach komputerowych*, 1998, `ftp://ftp.mimuw.edu.pl/pub/polszczyzna/ogonki/`
- Janusz S. Bień *Kodowanie tekstów polskich w systemach komputerowych*, 1999, `ftp://ftp.mimuw.edu.pl/pub/polszczyzna/ogonki/katow98.ps`

- Unicode `http://www.unicode.org`
- ECMA `http://www.ecma.ch`
- RFC (Internet Requests For Comments) `ftp://ds.internic.net/rfc`