Unix: internacjonalizacja i lokalizacja programów

Witold Paluszyński
witold.paluszynski@pwr.wroc.pl
http://kcir.pwr.wroc.pl/~witold/

Niniejszy dokument zawiera materiały do wykładu na temat koncepcji i narzędzi do internacjonalizacji oprogramowania w systemie Unix. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Wprowadzenie

Programy tworzone z myślą o rozpowszechnianiu ich na świecie muszą w jakimś stopniu obsługiwać języki obce, i związane z tymi językami konwencje notacyjne. Proces przekształcania programu na wersję międzynarodową, obsługującą więcej niż jeden język, nazywamy **internacjonalizacją**, oznaczaną skrótem `i18n`, co jest skrótem słowa *internationalization* (jakoby skraca 18 liter w środku słowa).

Program poddany internacjonalizacji musi następnie, po zainstalowaniu go na docelowym komputerze, wybrać i zdecydować się na swój język pracy. Jest to nazywane **lokalizacją**, w skrócie `l11n`.

Dobra wiadomość jest taka, że istnieją narzędzia wspomagające zarówno internacjonalizację jak i lokalizację programów. Jednak naprawdę miła sytuacja polega na tym, że istnieją co najmniej dwie koncepcje procesów i18n i l11n, z dwoma niezgodnymi ze sobą standardami (GNU: gettext, X/Open: catgets), i podzielonymi obozami producentów wspierających jedną albo drugą koncepcję.

Koncepcja gettext

Jedną z koncepcji, przedstawioną w tym dokumencie, jest koncepcja GNU gettext. W największym skrócie polega ona na zbudowaniu katalogów komunikatów tekstowych w różnych językach, i wyborze komunikatu w konkretnym języku przez podanie w programie źródłowym oryginalnego komunikatu w języku angielskim.

```
#include <libintl.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  setlocale( LC_ALL, "" );
  bindtextdomain( "hello", "/usr/share/locale" );
  textdomain( "hello" );
  printf( gettext( "Hello, world!\n" ) );
  exit(0);
}
```

Kompilacja programu: `cc -o hello hello.c -lintl`

Kluczowe elementy użycia `gettext`

- Funkcja `setlocale` zdefiniowana w pliku nagłówkowym `<locale.h>` inicjalizuje lokalizację programu, lub jej wybraną część.
  Wywołana w tym przykładzie, inicjalizuje całą lokalizację na dokładnie zgodną z ustawieniami środowiska użytkownika.
- Funkcja `bindtextdomain` zdefiniowana w pliku nagłówkowym `<libintl.h>` definiuję dla tak zwanej **dziedziny** (ang. *domain*) bazowy katalog dyskowy zawierający drzewo podkatalogów translacji komunikatów na inne języki.takich bazowych katalogów zawierających zestawy translacji, zwane dziedzinami. Każdy program może mieć swoją własną dziedzinę, lub pakiet programów może korzystać ze wspólnej dziedziny.
  Katalog `/usr/share/locale` jest domyślnym systemowym katalogiem dziedzin translacji (zatem konkretne wywołanie zawarte w tym przykładzie można było pominąć).
- Funkcja `textdomain` również zdefiniowana w pliku nagłówkowym `<libintl.h>` wybiera dziedzinę dla bieżącego programu.
- Funkcja `gettext` (również `<libintl.h>`) zamienia komunikat będący jej argumentem na odpowiedni komunikat w wybranym języku.

Tłumaczenie komunikatów

Dalszym elementem internacjonalizacji programu jest stworzenie wszystkich niezbędnych tłumaczeń komunikatów tekstowych zawartych w programie. Pierwszym krokiem może być utworzenie wzorcowego pliku takich komunikatów, w czym pomocny jest program `xgettext`:

```
xgettext -d hello -s -o hello.pot hello.c
```

W wyniku tego wywołania powstaje plik `hello.pot` (`pot` — *portable object template*) zawierający posortowany (opcja `-s`) zestaw komunikatów. Ten plik można następnie skopiować na pliki konkretnych translacji, i rozpocząć pracę. Można również użyć programu pomocniczego `msginit`:

```
msginit -l pl_PL -o polski.po -i hello.pot
```

W tym wywołaniu opcje `-i` i `-o` określają plik wejściowy i wyjściowy, natomiast opcja `-l` określa lokalizację tworzonej translacji. Ta lokalizacja powinna istnieć na bieżącym komputerze. Program `msginit` dodaje do pliku translacji szereg komentarzy w dość sformalizowanym formacie, pozwalającym go automatycznie obsługiwać.

Istnieją edytory obsługujące ten format. Jednym z nich jest GNU Emacs w trybie `po`.

Po zakończeniu tłumaczenia wszystkich komunikatów, plik `.po` (*portable object*) musi zostać przekształcony na format binarny `.mo` (*machine object*):

```
msgfmt -c -v -o hello.mo polski.po
```

W tym wywołaniu opcja `-o` określa plik wynikowy, `-v` (*verbose*) obszerne komunikaty z programu, a `-c` dokładne sprawdzanie. Nazwa pliku (basename) musi być zgodna z dziedziną zadeklarowaną przez program.

Na koniec utworzony plik translacji musi zostać umieszczony we właściwym katalogu. Tym katalogiem jest katalog `LL/LC_MESSAGES` albo `LL_CC/LC_MESSAGES` gdzie `LL` oznacza dwuliterowy kod języka, a `CC` dwuliterowy kod kraju. Położenie tych katalogów jest oznaczane względem katalogu bazowego, z którego korzysta program. Dla naszego przykładowego programu należy więc przenieść utworzony binarny plik translacji do:

```
cp hello.mo /usr/share/locale/pl_PL/LC_MESSAGES
```

Dalsze zagadnienia

Komplikacja pojawia się w przypadku wprowadzania zmian w programie, jeśli wprowadzają one nowe komunikaty, bądź modyfikują dotychczasowe. Oczywiście ten sam proces można powtórzyć, tworząc wszystkie pliki od nowa i wklejając w nie edytorem komunikaty przetłumaczone już wcześniej, i odpowiednio je modyfikując. Jednak dla dużego programu, zawierającego tysiące komunikatów w wielu językach, jest to uciążliwy proces.

```
xgettext -d hello -s -o hello-new.pot hello.c
msgmerge -s -U polski.po hello-new.pot
```

Opcja `-U` powoduje aktualizację pliku, z wstawieniem pustych komunikatów w miejscu nowych komunikatów oryginalnych.

Następnie należy ponownie przekształcić plik `.po` na `.mo` i przenieść otrzymany plik do właściwego katalogu:

```
msgfmt -c -v -o hello.mo polski.po
cp hello.mo /usr/share/locale/pl_PL/LC_MESSAGES
```

Używanie tak przygotowanego oprogramowania wymaga ustawienia przez użytkownika zmiennej środowiskowej `LANG`:

```
setenv LANG pl_PL
```

Oczywiście, żeby wszystko działało poprawnie, system docelowy musi być poprawnie zainstalowany, z odpowiednią lokalizacją, katalogami komunikatów, a także czcionkami, itp.