Programowanie dwujęzyczne

Drugi język

Linux jest międzynarodowym fenomenem. Został zapoczątkowany przez programistę – Fina mówiącego po szwedzku, wspieranego przez porucznika mówiącego po walijsku oraz Brazylijczyka nadzorującego proces rozwoju jądra systemu. Jądro to jest rozwijane przez hakerów/programistów z całego świata. Dlaczego zatem całe oprogramowanie jest pisane w angielskiej wersji językowej? W tym miesiącu Steven Goodwin przyjrzał się wielojęzycznemu programowaniu i pakietowi gettext.

STEVEN GOODWIN

Język angielski ma w dzisiejszym świecie to samo znaczenie, jakie łacina miała setki lat temu. Nie jest to ani najbardziej ekspresywny język, ani najbardziej popularny. Jest on jednak najbardziej rozpowszechniony. Obecność resztek starego Imperium Brytyjskiego i ciągły wzrost znaczenia Ameryki zmuszają nas do używania języka angielskiego, aby nadążyć za resztą świata.

Komputery i Internet zwiększyły ten językowy przymus. W angielskiej wersji językowej powstaje znacznie więcej stron internetowych niż w jakiejkolwiek innej. Większość języków programowania wykorzystuje takie angielskie słowa jak: if i while, bez względu na narodowość projektantów. Większość oprogramowania wykorzystuje znaki zachęty i komunikaty błędów w języku angielskim.

Ustawienia regionalne

Ustawienia regionalne definiują zestaw danych. Każdy wspierany język ma taki własny zestaw. Ustawienia regionalne mogą opisywać sposób traktowania informacji: liczby powyżej 1000 mogą być oddzielane albo przecinkami albo kropkami, data może być zapisywana w konwencji dzień-miesiąc-rok lub miesiąc-dzień-rok. Informacje te nie są związane z językiem, dlatego korzystamy z terminu „regionalne". Składają się na niego język i specyfika kulturalna. Osobny katalog jest tworzony dla obu kategorii.

Istnieją standardowe funkcje do formatowania łańcuchów regionalnych. Na przykład strfmon i strftime formatują odpowiednio tekst dla pieniędzy i daty.

Linux przejmuje kontrolę nad wieloma różnymi systemami na całym świecie i dlatego tworzenie oprogramowania w jednym języku wydaje się ksenofobiczne. Dodanie możliwości zmiany języka (lub locale) do tworzonego przez Ciebie oprogramowania wcale nie jest trudnym celem do osiągnięcia. Staje się świadectwem Twojego zaangażowania w projekt i całą społeczność Open Source. Nawet jeśli sam nie potrafisz przetłumaczyć tekstu, możesz ułatwić pracę komuś innemu. Wykorzystaj wskazówki zawarte w tym artykule.

W kierunku języka japońskiego

GNU/Linux wykorzystuje technikę znaną jako locales do określenia wielu rzeczy: stosownego tłumaczenia tekstu, zestawu znaków wymaganego do przedstawienia alfabetu i specyficznych ustawień środowiskowych – sposobu przedstawienia liczb i dat. Każde z tych zagadnień należy do kategorii ustawienia regionalne. W tym artykule skupimy się na tłumaczeniu tekstu.

Zacznijmy od najprostszego programu jaki znamy, Witaj Świecie!. Będziemy kodować w języku C, ale te same techniki mogą być zastosowane w innych językach. Można przetestować ekwiwalent wykorzystując kod z ramki: W PHP.

Całkiem oczywiste jest, gdzie powinny być umieszczone poddane translacji łańcuchy znakowe. Podczas kompilacji nie wiemy, jaki będzie faktycznie użyty łańcuch znaków, ani jaki język zostanie użyty. Zapobiega to włączaniu przez nas wszystkich możliwych tłumaczeń bezpośrednio do programu. Zamiast tego musimy utworzyć katalogi każdego słowa i frazy wykorzystywanej przez nasz program i użyć pakietu gettext do działania jako słownik. Zamieni to nasze (najczęściej angielskie) słowa z prawidłową wersją w uruchomionym programie. To co jest „prawidłowe" zostanie określone przez konkretne ustawienia regionalne danego użytkownika.

Musimy zrobić dwie rzeczy:

1. Wypełnić kod źródłowy znacznikami typu: „daj mi właściwe zdania dla frazy XYX".
2. Zbudować słownik translacji dla każdego języka, który będziemy wykorzystywać.

Wypełnienie kodu źródłowego znacznikami jest bardzo prostym procesem. My, jako programiści, musimy sprawdzić każdą linię kodu i wskazać, które linie tekstu będą wymagały tłumaczenia. Możemy to zrobić wykorzystując specjalną funkcję (nazwaną, bez niespodzianek, gettext). Sprawdzi ona słownik i przekonwertuje nasz łańcuch do odpo-

Kategorie regionalne

Kategoria	Znaczenie
LC_COLLATE	Kolejność znaków w alfabecie narodowym
LC_CTYPE	Stosowany do definiowania znaków.
LC_MESSAGES	Przetłumaczony tekst. Przedmiot naszego artykułu.
LC_MONETARY	Format i symbole pieniędzy
LC_NUMERIC	Format i symbole liczb
LC_TIME	Format i symbole daty i danych

wiedniej wersji językowej.

```
printf(gettext('Witaj ➥
Świecie!\n'));
```

Funkcję tę można odnaleźć w pliku nagłówkowym libintl, dlatego musimy:

```
#include <libintl.h>
```

Przy kompilacji pod GNU/Linux nie są wymagane żadne dodatkowe biblioteki do konsolidacji programu. Słowo GNU należy tutaj podkreślić. Jest tak, ponieważ cechy międzynarodowe są włączone bezpośrednio do glibc. Użytkownicy innych, uniksowopodobnych systemów, mogą nie mieć aż tyle szczęścia. Jednakże bez folderu z wersjami językowymi tłumaczenia nie zostaną przeprowadzone. Nie ma to znaczenia dla uruchomienia programu, ponieważ w razie braku tłumaczenia zostanie użyty język wzorca – najczęściej angielski. Programiści C pewnie zauważą także, że metoda ta nie jest wszechstronna, ponieważ istnieje więcej niż jeden sposób do zadeklarowania łańcucha. Na razie nauczyliśmy się tylko jednej metody oznaczania łańcuchów do tłumaczenia, dlatego będziemy potrzebowali innego sposobu, aby poradzić sobie w przypadkach, gdy wywołanie funkcji gettext zwróci błąd składni np.:

```
char *pHello = 'Witaj, ➥
Świecie!\n';
```

Aby obejść ten problem musimy utworzyć makro, które włącza znacznik, ale nie ma niekorzystnego wpływu na składnię.

```
#define gettext_noop(String) ➥
String
...
char *pHello = gettext_noop('➥
Witaj, Świecie!\n');
```

Przed wyświetleniem łańcucha na ekranie musimy wywołać moduł tłumaczący w normalny sposób, tak jak poniżej:

```
printf (gettext (pHello));
```

Znaczniki te są wykorzystywane nie tylko wtedy, kiedy program jest uruchomiony, służą również do wskazania, który tekst potrzebuje tłumaczenia. Przyjrzyjmy się krótko narzędziu, które korzysta ze znaczników, aby samemu zbudować słownik tłumaczeń. Gdybyśmy tworzyli słownik ręcznie (dlaczego nie?!), znacznik gettext_noop byłby niepotrzebny.

Niektórzy programiści preferują zamiast dziewięcioliterowej nazwy makra nazwy jednoliterowe, takie jak znak podkreślenia. Również dlatego, że słowo gettext (i oba nawiasy) może spowodować w wielu przypadkach przekroczenie limitu 80 znaków. To nic niezwykłego.

```
#define _(str) gettext (str)
#define N_(str) gettext_➥
noop (str)
```

Standard GNU preferuje spację pomiędzy nazwą i nawiasem, pomimo to jest ona często pomijana.

Teraz możemy posunąć się dalej w tworzeniu naszego obcojęzycznego słownika.

Wywołania wiedeńskie

Tworzenie pliku, który zawiera wszystkie łańcuchy programu, nie jest aż tak czasochłonne jak się wydaje. Jest to bardzo proste zadanie i może być wykonane przy wykorzystaniu narzędzia zwanego xgettext. Jest to jeden z kilku przypadków, gdzie „x” nie stanowi części nazwy programu dla X Windows. Jest to skrót od angielskiego „extract” (wyodrębnij). Program ten przeszukuje plik źródłowy, wyszukując wywołanie funkcji gettext (lub gettext_noop) i umieszcza tekst w gotowym do tłumaczenia pliku (o rozszerzeniu .PO) znajdującym się w odpowiednim katalogu. Program rozumie składnię C oraz inne języki programowania (patrz ramka: xgettext: wspierane języki programowania). Jest w stanie wyodrębnić składnię wywołania funkcji ze zmiennych i komentarzy.

```
$ xgettext -d lm helloworld.c
$ tail -n 3 lm.po
#: helloworld.c:5
msgid 'Witaj Świecie!\n'
msgstr ''
```

xgettext: wspierane języki programowania

C, C++, ObjectiveC
PO
Python
Lisp, EmacsLisp
librep
Java
awk
YCP
Tcl
RST
Glade

Jak można zauważyć, każdy kawałek tekstu ma znacznik ID i ekwiwalentny łańcuch gotowy do przetłumaczenia. Łańcuch ten może przechowywać tłumaczenie tylko dla jednego wybranego języka, dlatego plik ten staje się wzorcem. Każdy tłumacz bierze jego kopię i tłumaczy jego treść w swoim ojczystym języku. Czasami rozszerzenie pliku PO jest zmieniane na POT dla rozróżnienia pomiędzy wzorcem a konkretnymi wersjami językowymi znajdującymi się w odpowiednim katalogu.

Należy zauważyć, że xgettext będzie poszukiwał nazwy gettext. Nie rozumie on jednak na tyle składni języka C (ani żadnego innego języka), by znać przedstawione powyżej techniki, takie jak #define _(str). Nie oznacza to jednak, że takie sztuczki nie mogą być stosowane. Przykładem są dwa popularne rozwiązania. Jednym z nich jest określenie znaku podkreślenia jako dodatkowego słowa kluczowego. Będzie to działało w taki sam sposób jak gettext.

```
$ xgettext -d lm -k_ ➥
helloworld.c
```

Alternatywnie można wykorzystać pre-procesor języka C (powodując rozwinięcie makra) przed uruchomieniem xgettext.

```
$ xgettext -C -d lm <(gcc -E ➥
helloworld.c)
```

W PHP

Pisanie wielojęzycznego oprogramowania w PHP nie różni się od tworzenia aplikacji w języku C. Funkcje mają nawet te same nazwy! Jednakże, kiedy kod jest uruchamiany jako część strony internetowej, bardziej właściwe może być określenie ustawień regionalnych explicite. Mogą one pochodzić ze zmiennej sesji lub ciasteczka na maszynie użytkownika.

```
<?php
setlocale(LC_ALL, "pl_PL");
textdomain("lm");
echo gettext("Witaj Świecie!\n");
?>
```

Wynik setlocale można także uzyskać przez użycie funkcji putenv

```
putenv ("LANG=pl");
#include <stdio.h>
int main(int argc, char *argv[])
{
  printf("Witaj Świecie!\n");
  return 0;
}
```

W tym przykładzie wykorzystujemy flagę -C, aby zaznaczyć, że rezultat potoku jest plikiem źródłowym C. Użytkownicy automake mają ułatwioną sprawę, ponieważ Makefile wygeneruje te pliki automatycznie.

Należy również zauważyć, że pliki zawierające komentarze wykorzystują znajomy znak #(hash). Komentarze te są stosowane na cztery różne sposoby i są określone przez symbol następujący natychmiast po znaku #, tak jak w Tabeli 1.

xgettext może również dodać komentarze do pliku PO, np. gdy uzna on, że łańcuchy mogą być wykorzystywane do specjalnego formatowania. Plik PO zawiera nagłówek, który sygnalizuje datę ostatniej modyfikacji i nazwę tłumacza, który ostatnio go edytował.

Mając już plik wzorzec, musimy stworzyć katalog dla obcojęzycznych tłumaczeń, czyli na przykład dla języka francuskiego.

Tour de France

Wykonujemy najpierw zwykłą kopię pliku wzorca, a następnie dodajemy odpowiednie francuskie słowa do każdego msgstr.

```
msgid „Hello World!\n"
msgstr „Bonjour, le monde"
```

Łańcuchy dodajemy albo modyfikując plik bezpośrednio albo wykorzystując jedno z dostępnych narzędzi. Tłumacze korzystający z Emacsa mają tą przewagę, że mogą wykorzystywać tryb PO. Ci, którzy preferują GUI, mogą wykorzystać poeditor.

Aby plik tekstowy mógł być wykorzystany przez nasz program Hello World, plik musi być przekonwertowany do przyjaznego maszynie formatu binarnego. Program, który to robi nazywa się msgfmt – generuje on pliki o rozszerzeniu .mo zamiast .po. Jest on optymalny przy dostępie do wybranych łańcuchów, ułatwia również znajdywanie błędów – patrz Listing 1.

Pierwsze ostrzeżenie przypomina nam, że nie zmieniliśmy jeszcze nagłówka informacji. Możemy to naprawić poprzez korektę linii wskazującej właściwą stronę kodową.

```
„Content-Type: ➥
text/plain; ➥
charset=ISO-8859-1\n"
```

Aby określić właściwy kod strony kodowej, można się odnieść do ramki: ISO 8859 lub do [1] w celu pogłębionej analizy. Informacja ta jest bardziej użyteczna dla tłumaczy niż dla programistów. Dodatkowe informacje można znaleźć w [2].

Błąd sam się łatwo naprawił. W większych programach byłby trudny do zauważenia przez programistę. Można również sprawdzić łańcuchy pod względem właściwej liczby (i typu) argumentów, wykorzystując opcję -c. Teraz możemy to sprawdzić!

Norweski las

Aby nasz program użył właściwego słownika językowego i aby zostać szczęśliwym użytkownikiem ustawień regionalnych, musimy dodać parę dalszych linii kodu. Są one proste i wspólne dla wszystkich tego typu programów.

```
#include <locale.h>
...
char *pPackage = „lm";
char *pDirectory = „locale";
...
setlocale (LC_ALL, „");
bindtextdomain (pPackage, ➥
pDirectory);
textdomain (pPackage);
```

Funkcja bindtextdomain określa główny katalog przetłumaczonych skatalogowanych plików domenowych, podczas gdy textdomain wymaga tylko określenia nazwy *pakietu* lub programu. Nasz pakiet nazywa się lm, ponieważ stworzyliśmy plik lm.mo. Należy być ostrożnym określając względną ścieżkę katalogu locale, aby przypadkiem nie zmienić nazwy katalogu. W takim przypadku ścieżka może okazać się niedostępna.

W katalogu, w którym będą znajdować się tłumaczenia, musimy stworzyć podkatalog locale i skopiować plik lm.mo do właściwej lokalizacji.

```
$ mkdir -p locale/fr/LC_MESSAGES
```

Korzystając z faktu, że pakiet w każdym języku ma nazwę lm.mo, wykorzystujemy nazwę katalogu do rozróżnienia francuskiego lm.mo od niemieckiego lm.mo. Nazwa ta została wybrana przez konwencjonalne kody językowe. Opisano to w [3]. Katalog nazwany LC_MESSAGES jest potrzebny, gdyż może występować duże zróżnicowanie ustawień środowiskowych. Mogą tam też istnieć katalogi wskazujące format daty i sposób prezentacji numerów. Patrz ramka: Ustawienia regionalne.

Teraz można uruchomić program (bez po-

ISO 8859

ISO	Strony kodowe
ISO 8859-1	zachodnia lub zachodnioeuropejska
ISO 8859-2	Centralna Europa lub wschodnioeuropejska
ISO 8859-3	południowoeuropejska lub maltańska (i esperanto)
ISO 8859-4	północnoeuropejska
ISO 8859-5	wschodnioeuropejska, cyrylica
ISO 8859-6	arabska
ISO 8859-7	grecka
ISO 8859-8	hebrajska
ISO 8859-9	turecka
ISO 8859-10	nordycka (sámi, inuit, islandzka)
ISO 8859-11	tajska
ISO 8859-12	(dawna celtycka, obecnie wycofana)
ISO 8859-13	bałtycka
ISO 8859-14	celtycka
ISO 8859-15	euro
ISO 8859-16	południowo-wschodnio europejska (zawiera symbol euro)

Listing 1: Znajdywanie błędów

```
$ msgfmt  lm.po
msgfmt: lm.po: warning: Charset "CHARSET" is not a portable encoding
name.
      Message conversion to user's charset might not work.
lm.po:19: `msgid' and `msgstr' entries do not both end with '\n'
msgfmt: found 1 fatal error
```

Tabela 1: Symbole następujące po

Symbol	Typ komentarza	Uwagi
. (kropka)	Automatyczny	Nie powinien być ręcznie modyfikowany
: (dwukropek)	Odniesienia	Plik i numer linii łańcucha
, (przecinek)	Flaga	Zwraca uwagę, że tłumaczenie jest np. niejasne
(spacja)	Użytkownika	Nie podlegają automatycznym modyfikacjom przy przetwarzaniu

trzeby rekompilacji), wykorzystując francuskie ustawienia regionalne i zacząć obserwować rezultaty.

```
$ LANG=fr_FR./hello
Bonjour, le monde
```

Aby permanentnie zmienić ustawienia regionalne, należy w zwykły sposób wyeksportować zmienną środowiskową np.:

```
$ export LANG=fr_FR
$./hello
Bonjour, le Monde
```

Jeżeli korzystasz wyłącznie z angielskojęzycznego systemu – może to nie zadziałać w związku z faktem, że nie ma francuskich ustawień środowiskowych w Twoim systemie (inne potencjalne problemy zostały opisane w [4]). Plik /etc/locale.gen precyzuje ustawienia regionalne wygenerowane na Twojej maszynie. Plik /usr/share/i18n/SUPPORTED będzie wskazywał, które z nich mogą być zainstalowane. Generowanie ustawień regionalnych wykonuje się następująco:

```
$ su
# musisz mieć uprawnienia roota
do wykonania poniższych
instrukcji:
Password:
# echo „fr_FR ISO-8859-1" > ➥
/etc/locale.gen
# locale-gen
Generating locales...
 fr_FR.ISO-8859-1... done
Generation complete.
```

Użytkownicy Debiana mogą użyć dpkg-reconfigure locales.

Możesz to sprawdzić, wykorzystując Twój własny program lub (jeśli myślisz, że błąd może wystąpić w Hello World!) jeden z wielojęzycznych narzędzi GNU, takich jak rm.

```
$ LANG=fr_FR rm this_wont_exist
rm: Ne peut enlever `this_wont➥
_exist': Aucun fichier ou ➥
répertoire de ce type
```

Aby udostępnić Twój słownik dla innych, powinieneś go zainstalować w globalnym repozytorium plików .mo w /usr/share/locale/ (lub w lokacji określonej przez zmienną środowiskową: TEXTDOMAINDIR). Katalog ten wykorzystuje tą samą hierarchię co powyżej. Zainstalowanie tłumaczenia w tym katalogu (wymaga to również uprawnień administratora) będzie oznaczać, że Twój kod nie wymaga już określania katalogu funkcją bindtextdomain. Parametrowi funkcji katalog można przypisać wartość równą NULL.

Rozumiejąc już techniczny proces wielojęzycznego programowania, przyjrzyjmy się kilku bardziej wyszukanym aspektom programowania.

Hiszpańskie oczy

Wiekszość twórców oprogramowania ma swoją metodę radzenia sobie z łańcuchami, ulubioną łańcuchową bibliotekę, własne metody dynamicznego łączenia łańcuchów, dodawania liczby mnogiej czy też budowania większych zdań z części składowych (jak z werbalnego Lego zautomatyzowanych kolejowych komunikatów). Omówimy teraz kilka tego typu metod, podkreślając problemy (i ich rozwiązania).

```
printf('Kasuję %d plik%s', ➥
iNum, iNum==1?'':'i');
```

Powyżej znajduje się typowy przykład tworzenia liczby mnogiej. Przypadek „jednego pliku" wymaga pojedynczego rzeczownika, podczas gdy wszystkie inne wykorzystują liczbę mnogą pliki. Tak jest w języku polskim i w angielskim. Nie wszystkie jednak języki stosują tę regułę. Przypadek „zero plików" może nie być liczbą mnogą, jak w języku francuskim. Mogą też istnieć osobne słowa dla „zera" albo jedno i drugie (tak jak te w rodzinie języków bałtyckich). Można to skompensować osobną funkcją ngettext. Ma ona dwa ID łańcuchów jako parametry (po jednym dla liczby pojedynczej i mnogiej) oraz numer. Numer jest wykorzystywany do określenia wersji łańcucha, która powinna być użyta do tłumaczenia.

```
printf(gettext('Kasuję %d ➥
plik', 'Kasuję %d plików', ➥
iNum), iNum);
```

Po natrafieniu na znacznik ngettext, program xgettext wygeneruje dwa ID łańcuchów w pliku .PO, gotowe dla tłumacza wraz ze specjalnym komentarzem c-format.

```
#: helloworld.c:32
#, c-format
msgid 'Kasuję %d plik'
msgid_plural 'Kasuję %d plików'
msgstr[0] ''
msgstr[1] ''
```

Nie wszystkie problemy można rozwiązać przy pomocy funkcji ngettext. W pewnym momencie natrafimy na problem, który ujawni się dopiero wtedy, gdy użyjemy dwóch lub więcej argumentów funkcji printf. Wymaga ona ściśle określonej kolejności słów. Nawet prosty angielskojęzyczny program i pomylenie %d z %s może spowodować naruszenie ochrony pamięci. Po przetłumaczeniu prostej frazy, takiej jak „Istnieje %d plików nazwanych %s", może się zdarzyć, że wynikowy tekst będzie następujący: „nazwanych %s istnieje %d plików". Co więcej, my programiści nie wiemy wszystkiego o każdym możliwym tłumaczeniu. To jest coś, czemu nie możemy przeciwdziałać. Bardziej subtelne problemy mogą zdarzyć się w zwrocie „Kopiując plik z %s do %s".

Są dwie metody rozwiązania problemu kolejności słów. Pierwszy wymaga od tłumacza zmodyfikowania sposobu sformułowania, tak aby argumenty zawsze pojawiały się we właściwym porządku. Komenda msgfmt może być wywoływana z parametrem -c, co spowoduje sprawdzenie pliku .PO. W rzeczywistości opcja ta wykonuje trzy niezależne testy: formatu (ten, którego potrzebujemy w tym przypadku), nagłówka (na obecność i treść nagłówka) i domeny (sprawdzanie problemów z dyrektywami domenowymi).

W przypadku drugiej, preferowanej metody ciężar rozwiązania opiera na programiście. Łańcuch formatujący musi zostać tak skorygowany, aby opisywał porządek parametrów. Wracając do powyższego przykładu kopiowania plików otrzymamy:

```
printf(ettext ('Kopiowanie ➥
pliku z %1$s do %2$s'), ➥
pSrc, pDest);
```

Specjalne specyfikatory formatu %1$s i %2$s są zarządzane przez kod printf w glibc. Warianty inne niż GNU mogą nie oferować tej funkcjonalności.

Po wyjaśnieniu problemu kolejności słów powinno być się świadomym, że konstruowanie łańcuchów podczas wykonywania programu to zły pomysł. Przedstawione dotychczas rozwiązania działają tylko wtedy, gdy cały łańcuch jest przekazany tłumaczowi. Należy za wszelką cenę unikać dzielenia tekstu na sekcje z wykorzystaniem strcat (lub podobnych). Tłumacz nie musi rozumieć kolejności zdań, ani nie ma możliwości zmiany ich znaczenia. Każdy łańcuch zawarty w katalogu musi mieć sens w momencie prezentacji.

```
/* Nie należy kodować w ten ➥
sposób!! */
```

```
strcpy('Kopiuję plik z ');
strcat(pSrc);
strcat(' do ');
strcat(pDest);
```

W niektórych aplikacjach najtrudniejszym słowem do tłumaczenia jest „the”!. W języku angielskim jedynym określonym rodzajnikiem jest „the”, w przeciwieństwie do Francuzów, Anglików, czy Hiszpanów, którzy posiadają ich więcej. W zależności od języka, mogą pojawiać nietypowe wersje dla rodzaju męskiego, żeńskiego, nijakiego bądź liczby mnogiej. Ten sam problem pojawia się z rodzajnikiem nieokreślonym „a” (z angielskiego). Normalnie te słowa dodaje się jako część właściwego tłumaczenia. Jednak już teraz powinniście wiedzieć, że dynamiczne tworzenie łańcuchów nie jest najlepszym pomysłem. W niektórych przypadkach bardzo kuszące byłoby oszczędzenie na ilości tłumaczeń, tak jak na Listing 2.

Powinniśmy zmodyfikować łańcuchy tak, aby czytane miały formę „katalogi” i „pliki”. Dzięki temu po przetłumaczeniu forma ich jest poprawna, niezależnie od rodzajnika. Można argumentować, że gdybyśmy mieli część programu, tą powyższą skróconą wersję dublujemy pracę tłumacza! Przykład mamy na Listingu 3.

To prawda. Dublujemy pracę! Na szczęście nakład naszej dodatkowej pracy jest minimalny w porównaniu do udręki, jaka czekałaby innego programistę przy modyfikacjach, bądź niespodzianek wynikających z dobijających błędów powstałych z użycia niewłaściwego rodzaju rzeczownika. Jest to warte naszego wysiłku.

China girl

Ostatni implementacyjny problem, jaki powinniśmy rozważyć, dotyczy estetyki. Chodzi o rozmieszczenie elementów na ekranie, menu i wykorzystanie punktów tabulacji. Program może bowiem ładnie wyglądać w angielskiej wersji językowej, jednak w momencie gdy zmieni się któreś ze słów, z góry określony układ strony załamie się. Niemieckie słowa są na przykład średnio o 50% dłuższe od ich angielskich odpowiedników. Istnieją dwie opcje: albo zignorować długość słów albo obejść problem odpowiednio kodując.

Większość (jeśli nie wszystkie) narzędzi linii poleceń nie przejmuje się szczególnie formatowaniem. Informacja jest funkcjonalna i jednolita, będąc odpowiednią do parsowania przez skrypty. Oprogramowanie GUI może jawnie umieścić tekst w dwóch kolumnach, aby się spodobać użytkownikowi końcowemu. Nie ma w tym nic złego! Niestety uruchamiając program w innej wersji językowej lewa kolumna może się nałożyć na tekst znajdujący się w prawej.

Aby uniknąć tego problemu, trzeba napisać nieco więcej kodu. Może to wymagać dostosowania pozycji prawej kolumny np. poprzez wyliczenie najdłuższego kawałka tekstu po lewej stronie lub zawijania tekstu. Można też zastosować przewijanie tekstu wewnątrz widocznego okienka (jak w XMMS). W takim przypadku może się zdarzyć, że nakładające się na siebie znaki zostaną obcięte. Tłumacz będzie musiał wybrać krótsze wersje. Zastosowane rozwiązanie będzie zależało od chęci i ilości pracy włożonej przez Ciebie i Twoich tłumaczy. W aplikacjach, których sprzedaż jest zależna od ich prezencji (jak gry), jest to konieczność.

Listing 2: Mniej do tłumaczenia

```
if (ygetfiletype(szFilename) == DIRECTORY)
      pFiletype = gettext ('katalog');
if (ygetfiletype(szFilename) == FILE)
      pFiletype = gettext ('plik');
printf (gettext ('%1$s jest %2$s'), szFilename, pFiletype);
```

Listing 3: Dublowanie pracy

```
if (ygetfiletype(szFilename) == DIRECTORY)
      pFiletype = gettext ('katalog');       /* te same łańcuchy co
poprzednio – czy to oznacza mniej pracy? */
if (ygetfiletype(szFilename) == FILE)
      pFiletype = gettext ('plik');
printf('%s: %s', szFilename, pFiletype);      /* nie potrzeba tłumacze-
nia tutaj */
```

Unicode

Wszystkie przykłady w tym artykule używają znaków ASCII. Obejmuje to większość zachodnich języków, ale zaniedbuje te zestawy znaków, które wymagają dwóch bajtów, jak chiński. Aby wspierać je w pełni, musimy wykorzystywać Unicode. To zaś wymaga znacznie większej ilości pracy. Nie można korzystać z podstawowego typu char, zamiast niego wykorzystujemy wchar_t. Ponadto zamiast większości dobrze znanych funkcji (jak sprintf) należy wykorzystać ich wide wersje np. swprintf.

Wiedeń

Wraz z rozwijającym się programem dodawanych będzie coraz więcej łańcuchów. Tłumaczenie od nowa całego programu jest oczywistą stratą czasu. Dlatego należy korzystać z narzędzia msgmerge. Wykorzystuje ono oryginalny językowy wzorzec (plik .PO, często przemianowany na .POT) oraz nowo stworzony katalog językowy do tworzenia nowego .PO.

Ten nowy plik zawiera wszystkie pierwotne tłumaczenia połączone z nowymi.

```
$ msgmerge start_plik_po.pot ➥
biezacy_jezyk_po.po >➥
nowy_jezyk_po.po
```

Metropolis

Mając pakiet gettext możemy tworzyć prawdziwie wielojęzyczne oprogramowanie, bez znajomości któregokolwiek z tych języków. Używanie osobnych wersji językowych programu pozwala na rozdzielenie tłumaczenia pomiędzy tych, którzy operują danymi językami, bez konieczności rekompilowania kodu. Daje to klarowny, łatwo poprawialny kawał dobrej programistycznej roboty.

Zostawiam was z tą myślą i żegnam się. Au revoir. Auf Wiedersehen. Adiós and Arrivederci. Do zobaczenia! ■

INFO

[1] ISO8859 Alphabet Soup: *http://wwwwbs.cs.tu-belin.de/user/czyborra/charsets/*

[2] Informacje o językach: *http://www.eki.ee/letter/*

[3] Kody językowe: *http://www.loc.gov/standards/iso639-2/langcodes.html*

[4] GNU gettext FAQ: *http://www.haible.de/bruno/gettext-FAQ.html#integrating_noop*