Internationalization 'nd Localization

mobytrick

12 settembre 2014

Far parlare un sistema operativo in una determina lingua è un aspetto che vale la pena scoprire. Ogni Nazione ha i propri modi ed i propri metodi per esprimere le date, le ore, la moneta e via discorrendo, derivanti dalla sua storia e dalle sue tradizioni. Tutte queste peculiarità nazionali sono state riunite in un meccanismo denominato *locales*. Tramite opportuni interventi su questi è possibile impostare il proprio PC secondo le usuali abitudini. Qualsiasi installazione Linux, anche minimale, non può prescindere dalla presenza o dal settaggio dei `locales` (in fase di istallazione, una delle prime domande poste è quale lingua si desidera usare). Si parla spesso di versione italiana di qualche distro: la versione nazionale è caratterizzata dal fatto che i `locales` opportuni sono già predisposti.

La tecnica adottata per il sistema operativo può essere usata anche a livello di applicazione utente. Si vuol scrivere un programma C che stampi delle stringhe. Stringhe che sono delle frasi di senso compiuto in una determinata lingua, ad esempio, italiano. L'obiettivo è quello di scrivere un programma corredandolo delle medesime scritte tradotte però in un'altra lingua. In base alle configurazioni dei `locales`, il programma provvede alla stampa delle stringhe opportune. Partiamo da un esempio concreto. Il programma sottostante, contenuto nel file `main.c`, estrae l'ora dall'orologio di macchina. In base a delle scelte codificate nel programma, stampa <buon giorno>, oppure <buona sera> e via discorrendo. Le stringhe appaiono così perché il programma è stato pensato per un utilizzatore italofono.

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
  int           ora;
```

```
  extern int    get_ora (void);

  ora = get_ora ();
  if (ora < 4 || ora > 22)
    (void) printf ("buona notte\n");
  else
    if (ora <= 12)
      (void) printf ("buon giorno\n");
    else
      if (ora < 18)
        (void) printf ("buon pomeriggio\n");
      else
        (void) printf ("buona sera\n");
  exit (EXIT_SUCCESS);
}
```

NB: il sorgente della funzione `get_ora` si trova a fine documento.

Arrangiando il programma in maniera appropriata, e tramite delle configurazioni esterne al programma stesso, le stringhe potranno apparire in un'altra lingua, posto che sia stata predisposta la loro traduzione. Non c'è nulla di magico/automatico perché le traduzioni sono a carico di una persona in carne ed ossa.

Passando ai prerequisiti di sistema, sono necessari:

1. la presenza del pacchetto `gettext`
2. la corretta configurazione dei cosiddetti `locales`

Per l'installazione dei pacchetti mancanti si ricorre ad `apt-get` (per le distro Debian e derivate) o comando equipollente.

La configurazione dei `locales` è visibile tramite il comando:

```
# locale -a
```

il cui output potrebbe essere:

```
C
en_US.utf8
POSIX
```

Nel caso non esista la configurazione per una determinata lingua si rimedia con:

```
# dpkg-reconfigure locales
```

e selezionando le coppie paese-lingua volute. Volendo installare i `locales` per tutte le lingue (opzione `all`) il completamento della configurazione richiede svariati minuti.
Per poter impostare una determinata lingua si tengano a mente i concetti di *internazionalizzazione* e *localizzazione*. Per internazionalizzazione si intende la predisposizione all'utilizzo della lingua usata in una determinata nazione. Ma non è sufficiente. Si deve far intervenire la localizzazione. Non esiste infatti la corrispondenza biunivoca tra paese e lingua. Un esempio per chiarire le idee: esiste un paese chiamato Svizzera, ma non esiste la corrispondente lingua. Ed è vero anche il contrario: esiste la lingua esperanto senza che esista il corrispondente paese. La localizzazione indica, appunto, la lingua da usare. Il termine inglese *internationalization* è composto da 20 caratteri. Spesso viene abbreviato in `i18n`: `i` è il primo carattere del termine ed `n` l'ultimo. In mezzo ne rimangono 18. Pure *localization* viene abbreviato in `l10n`, con procedimento analogo.

All'atto pratico, per predisporre gli opportuni `locales` si usa il comando:

```
# export LC_ALL=xx_XX
```

ove `xx` -2 lettere **minuscole**- identifica la lingua (localizzazione) e XX -due lettere **maiuscole**- indica il paese (internazionalizzazione). Per la codifica dei paesi si fa riferimento al codice `ISO 3166-1-alpha-2`. Per le lingue si fa riferimento al codice `ISO 639-1`. Ad esempio:

```
# export LC_ALL=it_IT.UTF-8
```

imposta i `locales` italiani, mentre

```
# export LC_ALL=ch_IT.UTF-8
```

imposta quelli italo-svizzeri. Attenzione: non sono previste tutte le combinazioni paese/lingua. (La gestione delle combinazioni è politica). Ad esempio, un polacco che si trasferisca in Portogallo non troverà la combinazione `pt_PL`. O sceglie il polacco oppure il portoghese.

Passando all'esempio proposto -stampa di una stringa in base all'ora- sono necessarie alcune modifiche. Per prima cosa si azzerano le definizioni di qualsiasi `locales` tramite lo statement:

```
setlocale (LC_ALL, "");
```

il prototipo della funzione è definito nello header `locale.h`, da includere. Il passo successivo consiste nel definire il *domain* su cui si opererà. Quello che in altri ambiti si chiama spesso progetto, qui assume la denominazione di `domain` (i retisti e quelli che hanno dimestichezza con il mondo Windows prestino attenzione). Di seguito si indica la directory ove si troveranno i file contenenti le traduzioni relative al progetto. Supposto che il `domain` si chiami `esempio` (supremo sforzo di fantasia, eh?) e che le traduzioni si trovino in `${HOME}/bin`, allora si inseriscono i seguenti due statement:

```
textdomain ("esempio");
bindtextdomain ("esempio", /home/utente/bin");
```

Gli argomenti delle due funzioni sono, anzi, devono essere delle costanti stringa e la variabile shell `${HOME}` deve essere risolta esplicitamente. I prototipi delle due funzioni si trovano nello header `libintl.h`, da includere. Esauriti questi preliminari, i richiami a `printf` vanno modificati nel seguente modo:

da	`(void) printf ("questa e' un a stringa\n");`
a	`(void) printf (gettext ("questa e' una stringa\n"));`

La stringa, anziché essere passata direttamente a `printf` per la stampa, viene passata a `gettext`. In fase di esecuzione questa funzione, in base al settaggio dei `locales`, recupera dai file del `domain` le stringhe tradotte e le passa a `printf` per la stampa. Procedimento un po' complesso, ma dal funzionamento garantito. Per evitare pasticci, i richiami a `printf` contenenti dei format vanno rimodulati in maniera da separare la parte testuale, sottoposta a traduzione, dal formato, che rimane invariato. Quindi:

da	`(void) printf ("risultato: %d\n", valore);`
a	`(void) printf (gettext ("risultato"));` `(void) printf (": %d\n", valore);`

Il programma riportato all'inizio è divenuto questo (in grigio gli interventi):

```
#include <libintl.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
  int           ora;
  extern int    get_ora (void);

  setlocale (LC_ALL, "");
  textdomain ("esempio");
  bindtextdomain ("esempio", "/home/utente/bin");
  ora = get_ora ();
  if (ora < 4 || ora > 22)
    (void) printf (gettext ("buona notte\n"));
  else
    if (ora <= 12)
  (void) printf (gettext ("buon giorno\n"));
    else
   if (ora < 18)
     (void) printf (gettext ("buon pomeriggio\n"));
   else
     (void) printf (gettext ("buona sera\n"));
    exit (EXIT_SUCCESS);
 }
```

Ora occorre costruire il file con le traduzioni. Ci si serve di `xgettext`, cui si passa in ingresso il file contenente le stringhe da tradurre. L'output viene dirottato su un file la cui estensione è `.po` (**p**ortable **o**bject). Riferendoci all'esempio proposto:

```
# xgettext main.c --output=tmp.po
```

`xgettext` non opera alcuna traduzione, ma crea un canovaccio -il file `.po`- su cui intervenire. Il file, sul cui nome c'è il solo vincolo dell'estensione, è liberamente consultabile con un editor; è il seguente:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR , YEAR.
```

```
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2011-01-06 14:37+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME \n"
"Language-Team: LANGUAGE \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: main.c:16
#, c-format
msgid "buona notte\n"
msgstr ""

#: main.c:19
#, c-forma
msgid "buon giorno\n"
msgstr ""

#: main.c:22
#, c-format
msgid "buon pomeriggio\n"
msgstr ""
#: main.c:24
#, c-format
msgid "buona sera\n"
msgstr ""
```

Gli interventi sono:

- le righe che contengono delle stringhe vuote (`msgstr ""`) vanno completate, riempiendole opportunamente con la traduzione del messaggio soprastante. Per questa operazione ci si affida alla propria conoscenza della lingua, all'immancabile amico oppure a vocabolari tradizionali oppure on-line

- togliere la riga contenente la parola `fuzzy`. `xgettext` costruisce un file che va modificato. Per essere sicuri di ciò, viene inserita questa riga che ha una mera funzione di sentinella. Va tolta. Almeno questa modifica si sa che è stata fatta
- le righe che iniziano con il carattere virgolette (`"`) possono essere tolte, perché ininfluenti. Se vengono lasciate, le parti in maiuscolo vanno sostituite in maniera opportuna. L'uniche righe che devono rimanere sono quelle che iniziano con `"Content"` ove la stringa `CHARSET` va sostituita con `UTF-8`

Modificato il file `tmp.po`, che ora assume questa forma (è mostrata quella usata per le traduzioni in tedesco; per farla breve sono stati tolti pure i commenti)

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
msgid "buona notte\n"
msgstr "Gute Nacht"
msgid "buon giorno\n"
msgstr "Guten Morgen\n"
msgid "buon pomeriggio\n"
msgstr "Guten Nachmittag\n"
msgid "buona sera\n"
msgstr "Guten Abend\n"
```

si tratta di effettuare l'ultimo passaggio che consiste nel creare l'effettivo file che verrà poi utilizzato per le traduzioni. Si usa il comando:

```
# msgfmt --vvvv --output=esempio.mo tmp.po
```

il nome dell'output è obbligatorio e corrisponde al `domain`. Pure -l'estensione `.mo` (per **m**essage **o**bject)- è obbligatoria (`-vvvv` è stato usato per avere la massima verbosità in fase di esecuzione. È una mia libera scelta). Il file `.mo` è *quasi* di testo, in quanto contiene diversi caratteri non stampabili. Per i curiosi ecco il contenuto, opportunamente depurato, del file usato per le traduzioni in tedesco:

```
buon pomeriggio
buona sera
buon giorno
buona notte
Content-Type: text-plain; charset=UTF-8
Guten Nachmittag
Guten Abend
Guten Morgen
Gute Nacht
```

Il file `.mo`, con maschera di protezione 444, va poi installato nel posto convenuto. Poiché nel secondo parametro di `bindtextdomain` si era stabilito di mettere tutto il materiale in `${HOME}/bin`, in tale directory devono essere creati due ulteriori livelli ed il file `.po` va installato lì in fondo

```
# mkdir -p ${HOME}/bin/de/LC_MESSAGE
# install -m 444 esempio.mo ${HOME}/bin/de/LC_MESSAGE
```

ove `de` rappresenta la sigla della lingua tedesca. Il file `.mo` ed il canovaccio (file `.po`) possono essere eliminati.

La trafila:

- produzione del canovaccio, tramite `xgettext`
- editing
- creazione del file `.mo` (tramite `msgfmt`)
- creazione di `${HOME}/bin/xx/LC_MESSAGES`
- installazione del file `.mo`
- eliminazione dei file di lavoro

va ripetuta per ciascuna delle lingue previste.

Siamo giunti al termine. Ora si tratta di vedere se dopo tutte le tribolazioni la cosa funziona! Partendo dai sorgenti C si costruisce l'eseguibile. Lo si lancia in esecuzione, ottenendo le stringhe in italiano. Modificando opportunamente la variabile di ambiente `LC_ALL` adeguandola alle lingue per le quali è stata fatta la traduzione si dovrebbero ottenere le traduzioni corrispondenti. Supposto di aver settato i `locales` in questa maniera:

```
C
en_US.utf8
it_IT.utf8
de_DE.utf8
POSIX
```

si può usare il seguente script (che supporremo trovarsi nel file `testlang`):

```
#!/bin/bash
set IT US DE
for i in it en de
  do
    country=${i}
    lang=$1
    LC_ALL=${country}_${lang}.utf-8
    export LC_ALL
    echo "$LC_ALL"
    ./esempio
    shift
  done
exit 0
```

Se lanciandolo in esecuzione tra le 13 e le 17 tramite:

```
# ./testlang
```

si ottiene il seguente output:

```
it_IT.utf-8 buon pomeriggio
en_US.utf-8 good afternoon
de_DE.utf-8 Guten Nachmittag
```

è la prova provata che il marchingegno funziona come preventivato.

Per completezza viene riportato il source della funzione `get_ora`:

```
#include <stdlib.h>
#include <time.h>

int get_ora (void)
{
  time_t        nowbin;
```

```
  struct tm     *nowstr;

  if (time (&nowbin) < 0)
    exit (EXIT_FAILURE);
  nowstr = localtime (&nowbin);
  return (nowstr->tm_hour);
}
```