National Language Support unter Linux

Linux- und Internet-Kongreß, 11./12. Mai, Berlin

Jochen Hein

(Jochen.Hein@Informatik.TU-Clausthal.De)

12. Mai 1995

Übersicht über den Vortrag

- Motivation für die Verwendung von National Language Support
- Einführung in die Funktionen und Möglichkeiten des NLS
- Auswirkungen für den Benutzer des Systems
- Möglichkeiten zur Konfiguration
- Programmierung unter Berücksichtigung von NLS
- Ausblick auf die Zukunft

Status heutiger UNIX-Systeme

UNIX wurde in den USA entwickelt, die anvisierte Benutzergruppe waren Amerikaner.

Dies ist unter UNIX deutlich zu spüren, da die verwendete Sprache meist Englisch ist und nur der 7-Bit Zeichensatz US-ASCII unterstützt wird.

Als Standard-Format für Datum und Zeit werden die US-amerikanischen Konventionen verwendet.

Bei abweichenden Konventionen muß jedes Programm einzeln angepaßt werden.

National Language Support

Diese und andere implizite Beschränkungen will man mit *National Language Support* (NLS) aufheben. In diesem Vortrag werden die vom POSIX.2-Standard definierten Methoden erläutert.

Die allgemeine Unterstützung von NLS bezeichnet man auch als *Internationalisierung* (I18N).

Die Übersetzung von Texten in eine spezielle Sprache bzw. die Anpassungen an einen speziellen Kulturkreis bezeichnet man als *Lokalisierung* (L10N).

Zur Lokalisierung müssen die Quelltexte eines Programmes nicht verändert werden, wenn die Internationalisierung durchgeführt wurde.

Implementierung des National Language Supports

Die Funktionen des National Language Supports werden in der C-Bibliothek implementiert.

Viele Benutzerprogramme müssen angepaßt werden, damit die Funktionen des NLS (korrekt) verwendet werden.

Es ist kein Kernel-Support notwendig, die gesamte Implementation erfolgt im User-Space.

Übersetzung von Texten

Die erste Überlegung ist, daß Meldungen und Dokumentationen übersetzt werden müssen. Hierzu gibt es z.B. den im X/Open Portability Guide 4 (XPG4) festgelegten Standard.

Um nicht für jede Übersetzung eines Programmes eine neue Version zu erzeugen, lagert man die Nachrichten in *Message-Kataloge* aus.

Programme geben dann nicht mehr fest einprogrammierte Meldungen aus, sondern lesen sie z.B. mit der Funktion `catgets(3)` aus dem entsprechenden Katalog.

Alt: `printf("Hello Word!\n");`

Neu: `printf(catgets(catfd,setnr,msgnr,"Hello Word!\n")`

Erstellen von Message-Katalogen

Hier als Beispiel der Beginn der Datei `errlist.m` aus der `libc`:

```
$set 1 #ErrorList
$ #0 Original Message:(Unknown error)
# Unbekannter Fehler

$ #1 Original Message:(Operation not permitted)
# Keine Berechtigung dazu
```

Die Übersetzung in das interne Format der `libc` erfolgt mit dem Befehl `gencat(1)`. Dabei kann eine Header-Datei mit symbolischen Namen erzeugt werden (hier `all.h`), wenn diese verwendet wurden.

```
.../libc-linux/nls/linux/German> gencat -new libc.cat *.m -h all.h
```

Aktivierung eines Message-Kataloges

Message-Kataloge werden im Verzeichnis `/usr/lib/locale/$LANG` (bzw. `/usr/local/lib/locale/$LANG`) gesucht.

```
No-MS-DOS:~> touch /
touch: /: Permission denied
No-MS-DOS:~> setenv LANG=de_DE.88591
No-MS-DOS:~> touch /
touch: /: Keine Berechtigung
```

Format eines Lokale-Namens: Sprache[_Land][.Zeichensatz][,Version]

Programmierung mit Message-Katalogen

Zu Beginn muß der Message-Katalog mit `catopen(3)` geöffnet am Ende des Programmes mit `catclose(3)` geschlossen werden.

```
setlocale (LC_MESSAGES, "");
catfd = catopen ("katalog", MCLoadBySet);
printf(catgets(catfd, setnr, msgnr, "Text"));
catclose(catfd);
```

Tutorial von Patrick D'Cruze:
Host: `sunsite.unc.edu`
Pfad: `/pub/Linux/utils/nls/catalogs/Incoming`
Datei: `locale-tutorial-0.8.txt.gz`

Andere Implementationen für Übersetzungen

Diese Implementation ist nicht besonders übersichtlich. Eine andere Methode ist die Verwendung der `gettext(3)`-Schnittstelle. Hier werden die Original-Nachrichten anstelle von Nummern als Schlüssel verwendet.

Ähnlich ist auch die Programmierschnittstelle, die vom GNU-Projekt verwendet wird (`glocale`). Leider werden die anderen Probleme, die NLS aufwirft, dort nicht berücksichtigt.

Unter X können übersetzte Texte z.T. in den X-Resourcen des Programmes abgespeichert werden.

man-Pages können mit der Version 1.4d des `man`-Programmes auch sprachabhängig im Pfad `/usr/man/$LANG/` gespeichert werden.

Message-Kataloge in Shell-Skripten

Auch in Shell-Skripten können Message-Kataloge verwendet werden, indem das Programm `dspmsg(1)` zum Auslesen der Nachrichten benutzt wird.

Es können alle Formatierungen von `printf(1)` verwendet werden.

Leider stehen hier die symbolischen Namen für Nachrichten nicht zur Verfügung.

```
#!/bin/sh
export LANG=de_DE.88591

dspmsg -s 1 messages 1 "Are you sure? "
```

Übersetzung von Texten 2

Bei der Übersetzung von Texten ist es oft notwendig, daß die Variablen in einer anderen Reihenfolge ausgegeben werden müssen.

Daher wurde der Format-String von `printf(3)` bzw. `scanf(3)` um die Position der auszugebenden Variable erweitert.

```
printf (format, weekday, month, day, hour, min);
```

Sprache	Format-String	Ausgabe
US-amerikanisch	`"%1$s, %2$s %3$d\n"`	`Sunday, July 3`
Deutsch	`"%1$s, %3$d. %2$s\n"`	`Sonntag, 3. Juli`

Kategorien im NLS - LC_MESSAGES Sprache und Antworten

Die Kategorie LC_MESSAGES legt fest, welcher Message-Katalog von Programmen verwendet wird. In der Regel wird man diesen Katalog im Verzeichnis /usr/lib/locale/$LC_MESSAGES bzw. $LANG finden.

Es werden auch die möglichen Antworten auf Ja/Nein-Fragen in Form von regulären Ausdrücken festgelegt. Verschiedene Systeme verwenden hier unterschiedliche (erweitere) reguläre Ausdrücke.

Antwort	Deutschland	USA
yesexpr	"^[jJyY][[:alpha:]]*"	"^[yY][[:alpha:]]*"
noexpr	"^[nN][[:alpha:]]*"	"^[nN][[:alpha:]]*"

Programmierung von Ja/Nein-Abfragen

Die Auswertung einer Benutzerantwort kann mit folgendem Code durchgeführt werden, wenn die Funktion `rpmatch`(3) zur Verfügung steht. Andernfalls müssen die regulären Ausdrücke mit den `regexp` Funktionen der `libc` selbst ausgewertet werden.

```
ans = rpmatch(str);
if (ans == 1)
  printf("You responded affirmatively\n");
else if (ans == 0)
  printf("You responded negatively\n");
else
  printf("Your answer did not match\n");
```

Implementation der Funktion `rpmatch(3)`

```
int rpmatch(const char *response)
{
  regex_t re;
  int result;
  result = regcomp (&re, nl_langinfo(YESEXPR), REG_EXTENDED);
  if (result != 0) {                    /* Fehler */
      char errbuf[BUFSIZ];
      (void) regerror (result, &re, errbuf, BUFSIZ);
      printf("%s\n",errbuf); }
  if (regexec (&re,response,0, NULL, 0) == 0)
      return 1;                         /* Ja-Antwort */
...
}
```

Shell-Skripte mit NLS und Message-Katalogen

```
#!/bin/bash
dspmsg -s 1 messages 1 "Are you sure? "
read ans
if printf "%s\n" "$ans" | grep -Eq "$(locale yesexpr)" ; then
  dspmsg -s 1 messages 2 "Yes, you are sure\n"
else
  if printf "%s\n" "$ans" | grep -Eq "$(locale noexpr)" ; then
    dspmsg -s 1 messages 3 "No, you aren't sure\n"
  else
    dspmsg -s 1 messages 4 "Hm, what did you say?\n"
  fi
fi
```

Kategorien im NLS - `LC_CTYPE` Zeichensätze

In Europa werden die ISO-8859-*-Zeichensätze verwendet. Diese enthalten auch nationale Sonderzeichen wie z.B. Umlaute. In Deutschland wird der ISO-8859-1-Zeichensatz (auch bekannt als ISO-Latin-1) verwendet.

Bisher müssen viele Programme speziell konfiguriert werden, um diesen Zeichensatz zu benutzen. Einige Programme verwenden die Einstellung `LC_CTYPE=ISO-8859-1`.

Besonders problematisch sind Programme, die das 8. Bit intern verwenden. Ebenfalls problematisch sind Programme, die das 8. Bit ignorieren oder diese Zeichen als Sonderzeichen betrachten.

Kategorien im NLS - `LC_CTYPE` Zeichensätze 2

NLS definiert eine Reihe von Zeichenklassen, die die Zuordnung von Zeichen zu bestimmten Gruppen von Zeichen regeln. Die entsprechenden C-Funktionen sollten verwendet werden, anstatt das Rad neu zu erfinden.

Außerdem können die Funktionen `toupper(3)` und `tolower(3)` beeinflußt werden, so daß diese Funktionen auch für Umlaute korrekt arbeiten. Das „ß“ ist jedoch nicht umsetzbar, da es in Großbuchstaben durch „SS“ dargestellt werden soll.

Es werden auch Funktionen für Multi-Byte und Wide Characters definiert.

Kategorien im NLS - LC_CTYPE Zeichensätze 3

Zeichenklasse	Bedeutung	C-Funktion
`upper`	Großbuchstaben	`isupper()`
`lower`	Kleinbuchstaben	`islower()`
`alpha`	Alle Buchstaben	`isalpha()`
`digit`	Ziffern	`isdigit()`
`space`	Whitespaces	`isspace()`
`cntrl`	Control-Zeichen	`iscntrl()`
`punct`	Trennsymbole	`ispunct()`
`graph`	Druckbare Zeichen	`isgraph()`
`print`	Druckbare Zeichen	`isprint()`
`blank`	Leerzeichen (Space und Tab)	`isblank()`
`xdigit`	Hexadezimale Ziffern	`isxdigit()`

Die Zeichenklasse graph enthält nicht die space-Klasse, die Zeichenklasse print enthält jedoch die space-Klasse.

Die C-Funktionen sind im Header-File ctype.h definiert.

Kategorien im NLS - LC_COLLATE Sortierung

Je nach Nation existieren unterschiedliche Konventionen, wie Buchstaben und Zeichenketten sortiert werden sollen.

Im Deutschen werden z.B. Umlaute direkt zu den zugehörigen Vokalen zugeordnet, in skandinavischen Ländern werden sie am Ende einsortiert.

Die C-Funktionen `strcoll(3)` und `strxfrm(3)` beachten diese Regeln, die Funktion `strcmp(3)` jedoch nicht.

Diese Funktionen sind (noch) nicht korrekt implementiert.

Kategorien im NLS - LC_NUMERIC Formatierung von Zahlen

In verschiedenen Ländern werden auch unterschiedliche Regeln zur Formatierung von Zahlen verwendet.

Mögliche Unterschiede sind verschiedene Trennsymbole für die Tausender-Trennung und das Dezimal-Komma, sowie andere Vorschriften für die Anordnung der Trennsymbole (d.h. keine Tausender-Trennung, sondern eine beliebige andere).

Feld	Deutschland	USA	POSIX
Dezimal-Komma	Komma	Punkt	Punkt
Tausender-Trennung	Punkt	Komma	Komma
Gruppierung	"3;0"	"3;0"	""

Programmierung - Zahlen formatieren

Für die Ein- und Ausgabe von Zahlen wurden die Formatierungen für `printf(3)` und `scanf(3)` um das Zeichen ’ erweitert, mit dem die aktuelle Lokale verwendet wird.

Funktionsaufruf	Ausgabe
`printf("%d\n", 12345678);`	12345678
`printf("%'d\n", 12345678);`	12.345.678
`printf("%f\n", 123456.78);`	123456,780000
`printf("%'f\n", 123456.78);`	123.456,780000
`printf("%1$d:%2$.*3$d\n",12, 34, 2);`	12:34

Kategorien im NLS - LC_MONETARY Formatierung von Geldbeträgen

Jedes Land hat unterschiedliche Währungssymbole, die die Währung in Texten oder Ausgaben identifizieren. Es werden auch unterschiedliche Trennsymbole als Dezimal-Komma oder Tausender-Trennung verwendet.

Feld	Deutschland	USA
Dezimal-Komma	Komma	Punkt
Tausender-Trennung	Punkt	Komma
Währungssymbol	DM	$
Internationales Währungssymbol	DEM	US$

Es gibt noch eine Reihe weiterer Felder für diese Kategorie. Dort wird u.a. die Position von Vorzeichen festgelegt.

Programmierung 3 - Geldbeträge formatieren

Die Funktion `localeconv(3)` dient zum Zugriff auf die Formatierungsvorschriften in der Struktur `lconv` (siehe auch `locale.h`). Mehr Unterstützung, wie z.B. Funktionen zur Formatierung, findet man in der `libc` nicht.

In der Struktur `lconv` findet man nur die Werte, die Zahlen und Geldbeträge betreffen, und hat keinen Zugriff auf andere Lokale-bezogene Daten.

Kategorien im NLS - LC_TIME Formatierung von Zeit und Datum

Zeiten und Daten werden in verschiedenen Ländern unterschiedlich formatiert. Die Reihenfolge der Tage, Monate und Jahre in der Kurzform eines Datums variiert von Land zu Land.

Land	Datum
Deutschland	01.03.1995
Norwegen	01-03-1995
Griechenland	01/03/1995
USA	03/01/1995

Weiterhin sind die Namen der Tage und Monate zu übersetzen, sowie die entsprechenden Kurzformen mit bis zu drei Buchstaben.

Sprachabhängiges Programm `date(1)`

```
No-MS-DOS:~> setenv LC_TIME C
No-MS-DOS:~> date +"%A, %d. %B %Y, %X %Z"
Monday, 05. December 1994, 21:33:14 MET
No-MS-DOS:~> setenv LC_TIME de_DE.88591
No-MS-DOS:~> date +"%A, %d. %B %Y, %X %Z"
Montag, 05. Dezember 1994, 21:33:26 MET
```

Probleme treten auf, wenn z.B. in Shell-Skripten ein Programm die Lokale beachtet, ein anderes aber nicht.

```
date > dump.date.new
tar --newer ‘cat dump.date‘
```

Kategorien im NLS - LC_TIME Formatierung von Zeit und Datum

Nur die C-Funktion `strftime(3)` beachtet die eingestellte Lokale. Die Funktionen `ctime(3)`, `asctime(3)` und `localtime(3)` verwenden immer die Standard-Lokale.

Die entsprechenden Werte für Monats- und Tagesnamen bzw. deren Abkürzungen können mittels der Funktion `nl_langinfo(3)` zur direkten Verwendung ausgelesen werden. Das zugehörige Header-File ist `langinfo.h`.

```
printf("%s\n",nl_langinfo(ABDAY_1));
```

Verwenden von NLS in Programmen

Der POSIX-Standard legt fest, daß Programme die C- oder POSIX-Lokale benutzen, wenn keine andere Lokale angewählt wird.

Die C- oder POSIX-Lokale ist durch den ANSI-C-Standard festgelegt.

Programme schalten mit dem Befehl `setlocale(LC_ALL,"")` auf die vom Benutzer eingestellte Lokale um. Es ist auch möglich, einzelne Kategorien speziell zu verändern: `setlocale(LC_TIME,"C")`

Die aktuell verwendete Lokale kann mit `setlocale(LC_TIME,NULL)` abgefragt werden.

Mögliche Benutzereinstellungen

Jeder Benutzer kann über die Umgebungsvariable `LANG` die für ihn passende Lokale wählen, z.B. mit `setenv LANG de_DE.88591`. Diese Variable bestimmt grundsätzlich die Lokale.

Einzelne Kategorien können jedoch eine andere Lokale benutzen, indem z.B. `setenv LC_CTYPE ISO-8859-1` verwendet wird.

Alle auf diese Art eingestellten Kategorien können mit der Umgebungsvariablen `LC_ALL` überschrieben werden.

Eine Übersicht über die Auswahl der Lokale zeigt das Programm `locale(1)` an, wenn es ohne Parameter aufgerufen wird.

Übersicht über alle Kategorien und Einstellungen

Variable	Wirkung
`LANG`	Grundsätzliche Wahl der Lokale
`LC_COLLATE`	Sortierreihenfolge von Strings
`LC_CTYPE`	Zeichenklassen
`LC_MONETARY`	Geldbeträge
`LC_NUMERIC`	Numerische Formatierung
`LC_TIME`	Datum und Uhrzeit
`LC_MESSAGES`	Sprache für Meldungen und Abfragen
`LC_RESPONSE`	obsolet
`LC_ALL`	Überschreiben von allen Kategorien
`NLSPATH`	Suchpfad nach Message-Katalogen

Das Programm `locale(1)` zur Analyse von Lokalen

```
No-MS-DOS:~> locale
LANG=de_DE.88591
LC_COLLATE="de_DE.88591"
LC_CTYPE=ISO-8859-1
LC_MONETARY="de_DE.88591"
LC_NUMERIC="de_DE.88591"
LC_TIME="de_DE.88591"
LC_MESSAGES="de_DE.88591"
LC_RESPONSE="de_DE.88591"
LC_ALL=
No-MS-DOS:~> locale -ck decimal_point
LC_NUMERIC
decimal_point=","
```

Aufgaben des Systemadministrators

Der Systemadministrator muß unter Linux die entsprechenden Lokale-Definitionen erzeugen. Dazu dient das Programm `localedef(1)`. Die Syntax der Quell-Dateien ist in `charmap(5)` und `locale(5)` beschrieben.

Andere Systeme als Linux verwenden die Umgebungsvariable `LOCPATH`, um einen Suchpfad für Lokale-Definitionen einzurichten.

Beispiele für die `de`-Lokale findet man auf `ftp.tu-clausthal.de` im Verzeichnis `/pub/systems/Linux/SLT/nls`.

Diese Dateien und Programme sind kompatibel zur `libc`-Version 4, für die `libc`-Version 5 wurden diese Programme neu implementiert.

Aufgaben für Programmierer

In jedem Programm muß die Verwendung der aktuellen Lokale mit der Funktion `setlocale(3)` angefordert werden.

Es müssen die Funktionen verwendet werden, die die eingestellte Lokale beachten, es dürfen keine unangemessenen Annahmen (wie z.B. 7-bit Zeichen) gemacht werden.

Der Zugriff auf die einzelnen Werte der Lokale wird durch den POSIX-Standard nicht definiert. XPG4 definiert die Funktion `nl_langinfo(3)`.

Alle Informationen auf einen Blick

`locale-tutorial` von Patrick D'Cruze

man-Pages: `locale(1)`, `gencat(1)`, `dspmsg(1)`, `localedef(1)` `catgets(3)`, `catopen(3)`, `catclose(3)`, `nl_langinfo(3)`, `setlocale(3)`, `strftime(3)`, `locale(5)`, `charmap(5)`, `locale(7)`; Header-Files: `locale.h`, `langinfo.h`

O'Donnell, S.M.: Programming for the World: A Guide to Internationalization, Prentice Hall 1994

Lewine, D.: POSIX Programmer's Guide, O'Reilly, 1991

POSIX.2-Standard der IEEE.

Mailing-Liste: `li-international@li.org`,
`subscribe` im Body an `li-international-request@li.org`

Ausblick auf die Zukunft

Zunächst sind in allen Programmen die Aufrufe für `setlocale(3)` einzubauen. Anschließend muß das Programm auf Funktionsaufrufe untersucht werden, die nicht die Lokale-Einstellungen benutzen.

Diese Änderungen sollten natürlich in die nächste Version des Programmes einfließen.

Es sind noch Dokumentationen zu verfassen, damit alle Programmierer einfachen Zugang zu Lokalen haben.

Das GNU-Projekt arbeitet an der Internationalisierung seiner Programme.