Uluslararasılaştırma

16 Mart 2005

İçindekiler

1 Yerel Sistemi

Programların değişik ülke ve kültürlerin iletişim kurallarına uydurulabilmesi için geliştirilmiştir. Bu kurallar, tarih ve saatin yazım şekli gibi basitlerden, kullanılan dilin özellikleri gibi karmaşıklara kadar yayılabilir. Programlar çalışma ortamından kullanıcının belirlediği kural kümelerini alır ve kendilerini buna uydururlar. Bu kümelere yerel (locale) adı verilir.

1.1 Kategoriler

ISO C standardı aşağıdaki yerel kategorilerini tanımlar:

LC_CTYPE Karakter kodlamaları. Karakter sınıflama (isalpha, isspace, ...), karakter dönüştürme (tolower, mbstowcs, ...), sözcük karşılaştırma (strcmp, strcasecmp, ...) gibi fonksiyonların davranışını belirler.

LC_COLLATE Yerel kurallarına göre sıralama (strcoll, strxfrm, ...) fonksiyonlarının davranışını belirler.

LC_MESSAGES Program iletilerinin dilini belirler. 'gettext' seçili olan dili burdan bulur.

LC_MONETARY Para değerlerinin gösterimi için kullanılan kuruş ve binlik ayrım işareti, para birimi gibi bilgileri verir.

LC_NUMERIC Sayıların gösterim kurallarını verir. Biçimli G/Ç fonksiyonları (printf, atof, ...) bu kuralları kullanır.

LC_TIME Tarih ve saat gösterim biçimi, ay ve günlerin adları gibi bilgileri verir. strftime ve benzeri fonksiyonların davranışlarını belirler.

LC_ALL Tüm kategorileri kapsar.

Sözcüklerin karşılaştırılması, büyük/küçük harf dönüşümü, biçimli G/Ç gibi standart C kitaplığınca sağlanan görevler, o anda geçerli yerel seçimine göre otomatik olarak doğru sonuçları döndürürler. Yerel kurallarına göre sıralama, iletilerin çevrilebilmesi gibi özellikler için ise, programın bu özelliği destekleyecek biçimde yazılması gerekir.

1.2 Yerel İsimleri

Öntanımlı olarak her sistemde bulunan üç adet yerel adı vardır:

"C" Fonksiyonlar ISO C tarafından standartlaştırıldığı biçimde davranırlar. Programlar çalışmaya her zaman bu yerel ile başlar.

“POSIX” Posix standardını belirtir. C ile eş özellikler taşır.

“” Boş bir isim verildiğinde, ortam değişkenlerinden kullanıcının ayarladığı yerel seçilir.

Diğer yerel isimleri sistemden sisteme değişmektedir. Kabuk ortamında

locale –a

komutunu vererek kurulu yerelleri görebilirsiniz. Yerel isimleri genellikle şu biçimdedir:

dil [_bölge] [.karakterseti] [@değiştirici]

Köşeli parantez içindeki kısımlar isteğe bağlıdır. Dil için ISO 639 tarafından tanımlanmış iki küçük harften oluşan dil kodları (en, tr, fr, ...) kullanılır. Bölge ise ISO 3166 ile tanımlanan iki büyük harfle (US, GB, TR, ...) gösterilir.

1.3 Yerel Seçimi

Program içinden yerel seçmek için

```
char *setlocale (int category, const char *locale);
```

çağrısı kullanılır. İlk parametre değiştirilecek kategoriyi belirten ve yerel kategorileri ile aynı adı taşıyan makrolardan biridir. İkinci parametre ise seçilecek yerel adıdır. Fonksiyon geçerli yerel adını geriye döndürür. Dönen değer C kitaplığına ait olduğu için üzerinde değişiklik yapmayın, yapmanız gerekiyorsa yada ilerde kullanacaksanız bir kopyasını alın. Verdiğiniz yerele geçiş yapılamazsa geriye NULL değeri döner. İkinci parametre olarak NULL değerini vererek, o anki yerel adını bir değişikliğe yol açmadan öğrenebilirsiniz.

Program başlangıcında kullanıcının ayarlarını geçerli kılmak için:

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

komutunu vermelisiniz. Bir işlem için standard yerele ihtiyacınız olduğunda yada geçici olarak yerel değiştirmek istiyorsanız şöyle bir kod kullanabilirsiniz:

```
#include <locale.h>
char *old, *saved;
old = setlocale (LC_ALL, NULL);
if (old) {
```

```
    saved = strdup (old);
    set_locale (LC_ALL, "C");
    ...
    set_locale (LC_ALL, saved);
    free (saved);
}
```

2 UNICODE

UNICODE, Unicode Consortium organizasyonu tarafından geliştirilen ve her karaktere bir sayı değeri karşılığı atayan bir standarttır. Evrensel Karakter Seti (UCS) olarak bilinen ISO/IEC 10646 standardı ise, her iki organizasyonun işbirliği ile aynı sayısal karşılıkları taşımaktadır. Bu set,

- Yeryüzündeki tüm karakterlere bir sayı değeri atamayı amaçlamaktadır.
- Zaman içinde yeni karakterler eklenebilir ama eski karakterlerin sayı değerleri aynı kalır.
- Sayı değerleri UCS-4 adlı 31 bitlik set üzerinden verilir. İlk 7 bit 'Group', sonraki 8 bit 'Plane', sonraki 8 bit 'Row', en son 8 bit 'Cell' olarak gruplanır. İlk 'Plane' (group= 0, plane= 0) Basic Multilingual Plane (BMP) olarak adlandırılır. BMP, UCS-2 adı verilen 16 bitlik sete karşılık gelmektedir.
- UCS üzerindeki karakter kod noktaları genellikle **u**+0a31 biçiminde onaltılık sistemde sayılar olarak gösterilir.
- **u**+0021 – **u**+007e arasındaki kodlar ASCII ile, **u**+00a0 – **u**+00ff arasındaki kodlar ISO 8859-9 ile aynı tutulmuştur.

Unicode kodlarından oluşan karakter dizilerini (metinleri) bilgisayarda verimli bir biçimde saklayabilmek amacıyla çeşitli karakter kodlamaları (encoding) geliştirilmiştir.

2.1 UTF-8 (RFC 3629)

UTF-8 (Unicode Transformation Format - 8bit) kodlaması UNICODE karakterlerini 1-6 byte uzunluğunda diziler olarak kodlar. ASCII kodlaması içinde 0-127 arasında kalan karakterler aynen kendi kodları ile kullanılır, diğerleri ise byte dizileri haline gelir.

U+00000000 – U+0000007F	0xxxxxxx
U+00000080 – U+000007FF	110xxxxx 10xxxxxx
U+00000800 – U+0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+00010000 – U+001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U+00200000 – U+03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U+04000000 – U+7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

UTF-8 şu özellikleri taşır:

- Her saf (0-127 arası karakterlerden oluşan) ASCII dizisi geçerli bir UTF-8 dizisidir.
- 0 ile 127 arası değerler kendi karakter karşılıkları dışında dizilerde geçmezler. Böylece örneğin '%', ':', '(', ')' gibi karakterleri parse edip, diğer karakterleri aynen geçiren parser ve programlar (ayrıca C printf benzeri fonksiyonları) diğer UNICODE karakterlerinden etkilenmezler.
- Sıfır değerli byte dizi içinde geçmez. Böylece strlen gibi fonksiyonlar çalışmaya devam eder. Ancak gerçek karakter uzunluğu yerine byte uzunluğu döndürmeye devam ettiklerine dikkat edilmelidir.
- Bir UTF-8 byte'ı bir karakterin kod dizisinin ilk byte'ı ise, kendisinden sonra kaç byte geleceği hemen anlaşılır. Herhangi bir byte'ın bir karakter kodunun dizisine ait olduğu tek bir bit kontrolü ile anlaşılır.
- Boyer-Moore hızlı metin arama algoritması UTF-8 ile kullanılabilir.
- UTF-8 <-> Unicode çevrimleri kolaydır.
- UTF-8 dizilerini başka bir karakter kodlaması bilgisi olmaması durumunda istatistiksel olarak tesbit etmek kolaydır.
- MIME kodlamalarında ve başka yerlerde 'UTF-8' biçiminde yazılır.
- UTF-8 kullanan uygulamalar, güvenlik açısından her karakter dizisinin geçerliliğini kontrol etmelidir. Her olası byte dizisi geçerli bir UTF-8 dizisi değildir. Örnek olarak geçersiz UTF-8 dizisi 0xC0 0x80, eğer normal UTF-8 çevrimi işlemine sokulursa sıfır karakter değerini verir. Aynı şekilde 0x2F 0x2E 0x2E 0x2F (/../) yi yasaklayan bir program, geçersiz 0x2F 0xC0 0xAE 0x2E 0x2F dizisi ile kandırılabilir. Bu sebepten dışardan gelen her türlü UTF-8 dizisinin önce geçerliliği kontrol edilmelidir.

UTF-8 geriye uyumluluk sağlaması ve Türkçe için en az bellek israfına yol açması nedeniyle Uludağ projesinde öntanımlı karakter kodlaması olarak tercih edilmiştir.

2.2 UTF-16 (RFC 2781)

UTF-16 (Unicode Transformation Format - 16bit), Unicode BMP içinde kalan karakterleri 2 byte'lık diziler olarak kodlar. BMP dışında kalan karakterleri kodlayabilmek için surrogate pair (vekil çift?) denen 4 byte'lık diziler kullanılır.

U+00000000 – U+0000FFFF	xxxxxxxx xxxxxxxx
U+00010000 – U+0010FFFF	110110xx xxxxxxxx 110111xx xxxxxxxx

Çiftin üst on bitini içeren kısmı 0xD800 ile 0xDBFF arasındayken, alt on biti taşıyan kısım 0xDC00 ile 0xDFFF arasındadır. Bu değerler arasındaki 16 bitlik değerler Unicode üzerinde özel olarak UTF-16'ya ayrılmış ve hiç bir karaktere atanmamıştır.

İşlemciler 16 bitlik değerleri bellekte tutmak için farklı yöntemler kullanmaktadır. Örneğin 258 (0x0102) değeri little-endian işlemcilerde 0x02 0x01, big-endian işlemcilerde ise 0x01 0x02 diziliminde tutulmaktadır.

UTF-16 dizileri için bir endian standardı belirlenmemiştir. Metin UTF-16BE ya da UTF-16LE olarak işaretlenmemişse (MIME, vb ile) işleyici metnin ilk iki byte'ına bakar. Byte sıra işareti (Byte Order Mark, BOM) adlı karakter (0xFEFF) metnin byte sıralanmasını göstermek için ayrılmıştır. İlk iki byte 0xFF 0xFE ise metin little-endian, 0xFE 0xFF ise big-endian olarak işlenir. UTF-16 metinleri birbirine eklerken araya gelen BOM karakterini çıkartmaya dikkat edilmelidir. İşaretlenmemiş ve BOM taşımayan diziler için bir yol gösterilmemiştir.

3 Çeviri

Uygulamaların içerdiği iletilerin diğer dillere kolayca çevrilebilmesini sağlamak için "gettext" adında bir alt yapı geliştirilmiştir. Programların nasıl yazılması gerektiğine dair bir dizi kural, çeviri dosyaları için bir biçim, çevrilmiş iletilere erişmek için bir kitaplık ve çevirileri işlemek için çeşitli araçlardan oluşmaktadır. Uygulamaların en az değişiklikle çevrilebilir hale getirilebilmeleri için tasarlanmıştır ve çok yaygın olarak kullanılmaktadır.

Bir uygulamanın çevrilebilmesi için, kod ve derleme sistemi üzerinde gerekli değişiklikler programcı tarafından yapılmış olmalıdır. Bundan sonra çevirmen uygulamaya ait 'po' dosyalarını çevirerek uygulamayı yerelleştirebilir.

3.1 Kod Değişiklikleri

3.1.1 Temel Kullanım

'gettext' programlama arabirimi, Sun tarafından 1990 yılında Uniforum'a gönderilen bir öneriden yola çıkmıştır. Bugün için OpenI18N standardı tarafından belirlenir. Temel özelliği programların en az değişiklikle çevrilebilir hale getirilebilmesidir.

Burdaki çağrıları kullanabilmek için öncelikle 'libintl.h' adlı başlık dosyası include edilir. Bunu gettext çağrılarının kullanıldığı kod dosyalarında ve gettext ile çevrilen format string'lerini kullanan printf, sprintf, vb fonksiyonlarının bulunduğu dosyalarda yapmalısınız.

Çeviri sistemin en temel işlevleri; çevrilecek iletilerin bulunduğu alanı belirlemek, ve iletiye karşılık gelen çeviriyi almaktır. Tüm programların iletilerinin aynı alanda tutulması, bakım ve yönetim açısından zor olacağı için bu alan kavramı kullanılır. Çeviri alanı,

```
char *textdomain (const char *domain_name);
```

fonksiyonu ile belirlenir. domain_name seçilecek alan adını içeren bir C string'idir. NULL değerini taşıması durumunda seçimi değiştirmeden o an aktif olan alan adını öğrenebilirsiniz. Alanın dosya sisteminde hangi dizinden çevirilere bakacağını;

```
char *bindtextdomain (const char *domain_name,
        const char *dir_name);
```

ile belirleyebilirsiniz. Bu işlemler genellikle programın başında yapılır. Bir örnek:

```
setlocale (LC_ALL, NULL);
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);
```

Burada ilk önce, setlocale çağrısı ile standart C kitaplığının tüm kategorilerde kullanıcının seçtiği yerelleştirmeyi kullanması sağlanmakta. Daha sonra ise alana ait dizin seçilmekte ve alan etkinleştirilmektedir. PACKAGE ve LOCALEDIR makroları, autoconf sistemi tarafından "ornek" ve "/usr/share/locale" olarak ayarlanmıştır. Bu durumda bu satırlar çalıştığında Türkçe için /usr/share/locale/tr/LC_MESSAGES/ornek.mo" adlı dosyadan çeviriler yüklenecek ve kullanıma hazır hale gelecektir.

Bu alandan bir ileti çevirisi almak için ise,

```
char *gettext (const char *msgid);
```

fonksiyonu kullanılır. Dönüş değeri, verilen iletinin, o an seçili alan içinde, kullanıcının ayarladığı dile çevrilmiş halidir. Bu çağrıyı alanı değiştirip tekrar yaparsanız, ikinci sonuç yeni seçilen alandan gelir. Optimizasyon açısından döngü içinde kullanmamakta yarar görülebilirse de, GNU gettext alan değişmediği sürece çeviri sonuçlarını cache'lediğinden bu o kadar önemli değildir. Kullanıcının dilinde bir çeviri olmadığı durumlarda gettext, msgid iletisini aynen geri verir.

```
printf (gettext ("Hello dear translator, 6 * 9 = %d"), 42);
```

Örnekte görüldüğü gibi kullanılan yazı dizilerini gettext fonksiyonu içine almak gibi çok kolay bir işlem ile program çevrilebilir hale gelmekte. Çevrilecek iletiler başka bazı sistemlerdeki gibi özel ID ler ile belirtilmek yerine direk kendi metni ile aranmakta. Tabi bunu daha da kolaylaştırmak için C dilinin makro özelliğinden yararlanabiliriz:

```
#define _(Dizi) gettext (Dizi)
printf (_("Hello dear translator, 6 * 9 = %d"), 42);
```

Böylece her ileti başına yalnızca 3 karakter (altçizgi, parantez aç, parantez kapa) israf etmiş oluyoruz. Program yazmaya başlarken bir i18n.h başlık dosyası yaratıp içine bu makroyu

```
#define _(Dizi) Dizi
```

biçiminde koyarsanız ve iletilerinizi _(...) ile kuşatırsanız, ilerde programın çevrilmesi gerektiğinde tek yapmanız gereken bu i18n.h dosyasını değiştirip, programınızı gettext kitaplığı ile bağlamak olacaktır. Yeni programlarınıza başlarken çeviri alt yapısını bütünüyle eklemeseniz bile, bu basit işi yapıp, ilerde kolaylık sağlamanızı şiddetle öneriyoruz!

Programınızda bu değişiklikleri yaptıktan sonra çağıracağınız xgettext adlı program, tüm kodu tarayıp gettext ve _ ile işaretlenmiş metinleri PROGRAMADI.pot adlı bir dosyada toplar. Daha sonra çevirmenler bu özel biçimli dosyadan birer kopya çıkarıp içindeki metinleri çevirerek programınızı kolayca başka bir dile taşıyabilir.

3.1.2 Sorunlu Noktalar

Tabi metinler program içinde her zaman böyle bir fonksiyon çağrısı ile kolayca çevrilebilecek biçimde kullanılmıyor. Dillerin farklı yapıları da kimi sorunlar çıkarmakta. Örneğin isimlerin çoğul ve tekil hallerinin kullanımı dilden dile değişmekte.

```
printf ("%d file%s deleted", n, n == 1 ? "" : "s");
```

Burada son parametredeki ? deyimi n değeri 1 olmadığı koşullarda ’file’ yerine ’files’ şeklinde çıktı verilmesini sağlamakta. Ancak Türkçe ve benzeri dillerde İngilizce’deki bu kural yok. Bu durumlarda daha açık yazmayı tercih etmelisiniz:

```
if (n == 1)
    printf ("%d file deleted", n);
else
    printf ("%d files deleted", n);
```

Her zaman iletileri gettext ile kuşatmanız mümkün olmayabilir. Örneğin:

```
static const char *messages[] = {
    "some very meaningful message",
    "and another one"
};
fputs (gettext (messages[i]));
```

Burdaki sorun, basılacak olan ileti için gettext fonksiyonunun çağrılması, ancak çevrilecek iletileri içeren dosyayı oluşturacak xgettext programının kod içerisinde bu iletileri görememesidir. Bunu çözümü için;

```
#define gettext_noop(x) x
static const char *messages[] = {
    gettext_noop ("some very meaningful message"),
    gettext_noop ("and another one")
};
fputs (gettext (messages[i]));
```

kullanılabilir. gettext_noop için N_ makrosu da tanımlanabilir, böylece fazla yazı yazmaktan kurtulunur.

```
#define N_ (x) gettext_noop (x)
```

Bu iki makro da herhangi bir kod çağırmaz, yalnızca metinlerin çevrilecek metinler olduğunu xgettext'in anlamasını sağlarlar.

Çevirilerin rahatça yapılabilmesi için, özgün iletilerinizin belirsizlik içermeyen, anlaşılır, tam tümceler olmasına dikkat edin. Sözcükleri birleştirmek yerine birden fazla tümce kullanmayı tercih edin, örneğin:

```
printf ("File %s is %s", filename,
        size < 16000 ? "small" : "big");
```

yerine:

```
printf (size < 16000 ? _("File %s is small")
       : _("File %s is big"), filename);
```

Tek bir durumu anlatan çok satırlı iletileri, satır satır basmak yerine, içinde \n ile satır dönüşü içeren tek bir ileti olarak kullanmaya çalışın. Böylece çevirmenler yarım satırları çevirme ve bölünme yüzünden tutarsız çevirmeler yapma durumunda kalmayacaklardır.

3.1.3 İleri Kullanım

Tek bir çeviri alanı birçok program için yeterli olsa da, bunun yetersiz kaldığı bazı durumlar var. Örneğin bir dizi program aynı hata mesajlarını kullanıyorlarsa bunlar tek bir alanda toplanabilir, böylece bir kere çevrilmeleri yeterli olur. Kitaplıklarda ise fonksiyonların çağıran programın alanından bağımsız, kendi alanlarından çeviri almaları gereklidir. Her ne kadar bu sorun textdomain () çağrıları ile çözülebilirse de, yavaş ve kullanışsız olur. Bu amaç için:

```
char *dgettext (const char *domain_name, const char *msgid);
```

fonksiyonu yardımcı olur. Bununla farklı alanlardan ileti çevirileri alabilirsiniz.

gettext sistemi yalnızca ileti çevirilerini getirmekle kalmaz, aynı zamanda bunları saklandıkları karakter setinden, o anki yerelleştirmenin LC_CTYPE ile belirlenen etkin karakter setine de çevirir. Metinleri yerelleştirmeye bağlı kalmadan kullanabilen programlar, çevirileri UTF-8 karakter setinde isteyebilir. Bunun için,

```
char *bind_textdomain_charset (const char *domain_name,
        const char *codeset);
```

kullanılır. Örneğin:

```
bind_textdomain_charset (PACKAGE, "UTF-8");
```

gettext çeviri bulamadığı durumda verilen iletiyi aynen döndürdüğü için, bunun kullanıldığı durumlarda program kodundaki metinlerin UTF-8 olarak tutulması gerektiğine dikkat edin!

3.2 Derleme Sistemi Değişiklikleri

Kimi uygulamalar arşiv dosyasından açıldıklarında tek bir dizin içinde yayılır. Kimi uygulamaların ise kendi dizin hiyerarşisi mevcuttur. gettext kodu ve gereksindiği dosyalar çok sayıda ve karmaşık olduğu için, en azından onları uygun bir biçimde ayrı dizinler olarak koymanız tavsiye edilir. Uygulama GNU autoconf sistemini kullanıyorsa gettext entegrasyonu çok kolay olacağı için öncelikle uygulamayı autoconf standardına taşımayı düşünün.

Çerçeveyi otomatik olarak kurmak için “gexttextize” adlı uygulama kullanılır. Uygulamanın en üst dizininde

```
gettextize --intl
```

komutunu verdiğinizde aşağıdaki işlemleri yapar:

1. Yerelleştirmenin kullanımı ile ilgili bilgiler veren bir ABOUT-NLS dosyası en üst dizine kopyalanır.
2. Çevirileri tutacak bi ’po’ dizini yaratılır.

3. gettext kodlarını tutacak bir ’intl’ dizini yaratılır. Burdaki dosyalar kopyalanmak yerine sistemden sembolik link yapılır. Bu diskte az yer kullanarak tasarruf etmeyi ve dosyaların sistemde kurulu sürümleri ile güncel kalmasını sağlar. Uygulamayı arşivlerken tar’ın -h seçeneğini kullanarak dosyaların asıllarının arşivlenmesini sağlamalısınız. gettextize’nin --copy seçeneğini kullanarak dosyaların link edilmek yerine kopyalanmasını sağlayabilirsiniz. Bu yolu seçerseniz arada bir gettextize komutunu çağırarak dosyaların yeni sürümlerini uygulamanıza dahil etmeyi unutmayın.

4. Autoconf için config.rpath ve mkinstalldirs dosyaları ’configure’ destek dosyalarının olduğu dizine kopyalanır. Automake de kullanıyorsanız, gereken makro dosyaları ’m4’ dizinine kopyalanır.

Bu işlemden sonra bazı dosyalarda ufak değişikler yaparak entegrasyon tamamlanır:

3.2.1 ’po’ Dizini

Burda öncelikle POTFILES.in adlı bir dosya açıp içine çevrilecek iletileri içeren kod dosyalarının uygulamanın en üst dizinine göre göreceli yolunu yazmalısınız. Örnek bir dosya:

```
# List of source files containing translatable strings
lib/error.c
lib/getopt.c
src/main.c
src/ui.c
```

Bir sonraki adım LINGUAS adlı bir dosyaya hangi çevirilerin mevcut olduğunu boşluklarla ayrılmış biçimde yazmak. Kullanıcılar hangi dilleri istediklerini burayı değiştirmek yerine LINGUAS adlı ortam değişkenini değiştirerek ayarlayabilir. Almanca, Fransızca ve Türkçe çeviriler içeren bir uygulama için bir örnek:

```
# Available languages
de fr tr
```

Tabiki çeviri dosyalarını da de.po, fr.po, tr.po olarak buraya koymalısınız.

3.2.2 ’configure.in’ Dosyası

Bu dosya autoconf tarafından işlenip ’configure’ scriptini yaratmakta kullanılmaktadır. Yeni uygulamalarda ’configure.ac’ adını da taşıyabilir. gettext’in işleyebilmesi için burada şu işlemler yapılmalıdır:

1. PACKAGE ve VERSION değerlerini tanımlamalısınız. Eğer automake kullanıyorsanız

   ```
   AM_INIT_AUTOMAKE(myapp,1.0.5)
   ```

2. Uluslararasılaşma desteğini kontrol etmelisiniz.

   ```
   AM_GNU_GETTEXT
   ```

3. Değişiklerin yazılacağı dosyaları belirtmelisiniz.

   (a) Eski autoconf modeli için:

   ```
   AC_OUTPUT([ ...sizin dosyalarınız...
           intl/Makefile po/Makefile.in])
   ```

   (b) Yeni autoconf için:

   ```
   AC_CONFIG_FILES([
           ...sizin dosyalarınız...
           intl/Makefile
           po/Makefile.in
   ])
   ```

3.2.3 'Makefile.in' Dosyası

Derleme sırasında intl ve po dizinlerinizin de işlenmesini sağlamalısınız. Eğer automake kullanıyorsanız 'Makefile.am' dosyasında

```
SUBDIRS = intl ...sizin dizinleriniz... po
```

şeklinde bir değişiklik yeterli olacaktır.

3.2.4 Kod Dizinleri

Program girişinde çeviri alanınızı ayarlarken kullanacağınız LOCALEDIR makrosunu kod dizininiz içinde 'Makefile.in' yada automake kullanıyorsanız 'Makefile.am' dosyasında şöyle tanımlayabilirsiniz:

```
datadir = @datadir@
localedir = $(datadir)/locale
DEFS = -DLOCALEDIR=\''$(localedir)\'' @DEFS@
```

Programlar derlendikten sonra @LIBINTL@ (yada libtool kullanıyorsanız @LTLIBINTL@) ile bağlandıklarından emin olun. Bunu şöyle yapabilirsiniz:

```
LIBS = ...sizin kitaplıklarınız... @LIBINTL@
```

Derleyicinizin 'intl' dizinindeki C başlık dosyalarına erişebilmesi için, derleme hedeflerinizde

```
-I$(top_srcdir)/intl
```

parametresinin derleyiciye verilmesini sağlayın.

3.3 Çeviri

Çeviriye başlarken ilk önce, paketadı.pot adlı şablon dosyasından DİL.po (Türkçe için tr.po) adında bir kopya çıkartılır. Daha sonra bu dosyanın açıklamaları ve başlık bilgileri düzenlenir, ve iletiler teker teker çevrilir.

3.3.1 Açıklama

Örnek bir .pot dosyasının giriş kısmı:

```
# SOME DESCRIPTIVE TITLE.
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
```

Bu kısımda basitçe dosyanın ne olduğunu, paketle aynı lisansa sahip olduğunu, ve çeviren kişi yada kişilerin isim ve elektronik mektup adreslerini bildiriyoruz. Bir örnek:

```
# Turkish translation of Imposter
# This file is distributed under the same license as the Imposter package.
# Gurer Ozen <email@adres.com>, 2003.
```

3.3.2 Başlık

Örnek bir .pot dosyasının başlık kısmı:

```
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: imposter-devel@lists.sourceforge.net\n"
"POT-Creation-Date: 2004-05-30 23:50+0300\n"
"PO-Revision-Date: YEAR-MO-DA [HO]MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
```

```
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

Görüldüğü gibi burda çeviri ile ilgili daha detaylı bilgiler var. Açıklamalar dosyayı okuyan kişiye hitap ederken, bu kısım çeviri sistemi tarafından kullanılıyor. Örneğin kullanılan karakter seti burda belirleniyor. Burada PACKAGE ve VERSION yerine paket adı ve versiyonunu, PO-Revision-Date kısmında çeviriyi en son düzenlediğiniz tarihi (Yıl-Ay-Gün Saat:Dakika), Last-Translator'a adınız ve emailinizi, Language-Team'e ise çeviri grubunun bilgisini girmek yeterli. İstediğimiz karakter setini (mesela ISO-8859-9) kullanabilmekle birlikte (gettext programa bunları o anki geçerli karakter setine çevirip verecektir), sistemde genel olarak UTF-8 kullanacağımız için UTF-8 kullanmakta yarar var. Daha sonra anlatacağımız "fuzzy" bayrağını kaldırmalısınız.

Örnek bir tr.po dosyası başlığı:

```
#
msgid ""
msgstr ""
"Project-Id-Version: imposter 0.1\n"
"Report-Msgid-Bugs-To: imposter-dev@lists.sourceforge.net\n"
"POT-Creation-Date: 2004-05-30 23:50+0300\n"
"PO-Revision-Date: 2003-07-03 18:50+0300\n"
"Last-Translator: Gurer Ozen <email@adres.com>\n"
"Language-Team: tr <tr@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

3.3.3 İletiler

.pot dosyasından örnek bir ileti:

```
#: src/ui.c:84
msgid "Select a presentation file..."
msgstr ""
```

Çit işareti ile başlayan açıklama satırında çevrilecek iletinin kod içinde hangi dosyanın hangi satırından alındığı belirtilmekte. msgid çevrilecek iletiyi taşımakta. msgstr yi ise şimdi biz dolduracağız. İletinin tr.po içindeki çevrilmiş hali:

```
#: src/ui.c:84
msgid "Select a presentation file..."
msgstr "Bir sunum dosyası seçin..."
```

olacak. Bu şekilde bütün iletiler çevrildiğinde işimiz tamamlanmakta.

3.3.4 Özel durumlar

Programların yeni sürümleriyle birlikte içerdikleri iletiler de değişmektedir. Bazı yeni iletiler eklenirken, bazıları değişmekte, bazıları ise çıkarılmaktadır. gettext sistemi .po dosyalarını güncellerken yeni eklenen iletilerin çevirilerini boş bırakmaktadır, bunlar çevirmen tarafından doldurulur.

Çıkarılan iletiler,

```
#~ msgid "/File/_Close"
#~ msgstr "/Dosya/Kapa_t"
```

biçimine gelir. Bunları dosyadan çıkarabilirsiniz.

Değişen iletiler ya da gettext sisteminin düzgün çevrildiğine karar veremediği iletiler ise fuzzy olarak işaretlenir:

```
#: src/main.c:70
#, fuzzy
msgid "I don't have eny easter eggs... or do I?"
msgstr "Hiç sürprizim yok... yoksa var m??"
```

Burda özgün iletideki 'any' kelimesi 'eny' olarak değiştirildiğinde, gettext .po dosyalarında sorun olabileceğini belirtmek için iletinin çevirisini

```
#, fuzzy
```

satırı ile işaretlemiş. Bu durumda çevirmen çeviriyi kontrol eder. Gerekirse düzeltip fuzzy bayrağını kaldırır.

C programlarında karşılaşılabilecek bir başka özel çeviri durumu ise format stringleridir. Bunlar C tarafından basılacak çıktıyı şekle sokmak için kullanılır. Bir örnek:

```
#: src/file.c:167
#, c-format
msgid "cannot open file %s"
msgstr "%s dosyası açılamadı"
```

Burdaki c-format bayrağı, gettext sistemi tarafından otomatik olarak konur ve iletinin bir format stringi olduğunu söyler. İletinin format stringine benzemesi ama olmaması durumunda no-c-format bayrağı ile de karşılaşabilirsiniz. Format stringleri içindeki %s %d gibi % ile başlayan deyimler iletinin içind o yere başka bir değerin konacağını gösterir. Örnekte programın basacağı ileti örneğin "cannot open file resim.jpg" ya da Türkçe etkin iken "resim.jpg dosyası açılamadı" olacaktır.

Bu iletilerde % ifadelerini (%s: metin, %d: tamsayı, %f: sayı, %c: karakter, vs) korumalısınız. Aksi durumda program hatalı çalışacak ya da çakılacaktır. Dil yapısından dolayı yerlerini değiştirmeniz gereken durumlarda aşağıdaki gibi kullanın:

```
#: src/data.c:1203
#, c-format
msgid "%d of %d files processed"
msgstr "%2$d dosyadan %1$d adet dosya işlendi"
```

Burada özgün iletide ikinci sırada olan toplam dosya adedini başa almak için ’%’ ile ’d’ (ya da hangi tür değer basılıyorsa onun harfi) arasına SIRANO ve ’$’ koyarak iletiyi programı bozmadan değiştirmiş olduk.

4 Yazılımların Türkçe ile Sorunu Ne?

Pek çok yazılım Türkçe yerelleri (tr_TR ve tr_TR.UTF-8) ile kullanıldığında istenilmeyen sonuçlar veriyor veya hiç çalışmıyorlar. Gözlemlenen Türkçe sorunları büyük/küçük harf çevrimleri sırasında ortaya çıkıyor. Sorunun asıl kaynağı ise İngilizce’deki i,I karakterlerinin Türkçede farklı karşılıklarının olması.

İngilizce’de i karakterinin büyük harf karşılığı I olarak ifade ediliyor. Noktalı (küçük) ve noktasız (büyük) olmak üzere yalnızca iki i,I karakteri bulunuyor.

Türkçe’de ise i,İ,ı ve I karakterleri noktalılar ve noktasızlar olmak üzere dört farklı karakter bulunuyor. Sorunun kaynağı olarak, İngilizce’deki i<->I dönüşümü Türkçe’de i<->İ ve ı<->I olmak üzere farklı dönüşümler ile ifade ediliyor.

Herhangi bir yazılım i karakterini büyük harfe veya I karakterini küçük harfe çevirmek istediği zaman üç farklı sorundan biri ortaya çıkıyor.

1. Dönüşüm gerçekleşmiyor ve karakter eski hali ile yazdırılıyor. Yazılım dönüşümü yalnızca integer türü değerlerle çalışabilen toupper() fonksiyonu ile yapmaya çalışıyor ve dönüşüm gerçekleşmiyor.
2. Yazılım tek bayt ile ifade edilen İngilizce i,I karakterlerini çok baytlı İ,ı karakterlerine **aynı bellek alanında** dönüştürerek bellek alanını (dolayısı ile girdi metinini) bozuyor.
3. Yazılım anahtar kelimesini Türkçe yereli için doğru bir şekilde gerçekleştiriyor, fakat İngilizce dönüşümü göz önüne alınması gereken anahtar kelime bozulmuş oluyor.

Yukarıdaki sorun tipleri için farklı çözüm yöntemleri izlenmesi gerekiyor.

4.1 Sorun: toupper() ile karakter dönüşümü gerçekleşmiyor

Bu sorun tr_TR.UTF-8 yerelinde ortaya çıkıyor. Eğer toupper çağısı yapılıyorsa dönüş değerinin mutlaka kontrol edilmesi ve eğer dönüş değeri 128bit’den büyük ise

dönüşümün geniş karakterler ile çalışabilen towupper() ile yapılması gerekiyor. towupper() fonksiyonunu çağırmak için karakteri öncelikle çok-baytlı karakter dizisinden geniş karakter'e çevirmek gerekiyor.

Aslen tüm büyük/küçük harf dönüşümlerinin geniş karakterler üzerinden yapılması en ideal çözümken, yazılım geliştiriciler performans kaygıları nedeni ile karakterin niteliğine göre dönüşüm fonksiyonunu seçmeyi uygun görüyorlar. Fakat bu yöntem çoğu zaman (hemen hemen her zaman) Türkçe yerelinde hatalı çalışan uygulamaları doğuruyor.

4.2 Sorun: Aynı bellek alanında dönüşüm bellek alanını bozuyor

Sorun i,I tek baytlı karakterlerinin İ,ı çok baytlı karakterlerine dönüşümü sırasında meydana çıkıyor. Dönüşümü tek bir bellek alanı (buffer) üzerinde gerçekleştirmeye çalışan yazılım i,I karakterleri yerine yazılan İ,ı karakterlerinin son baytılarının bir sonraki karakterin üzerine yazılmasına neden oluyor.

Bu sorunun çözümü için dönüşümün yeni bir bellek alanı üzerinde yapılması ve boyut değişimlerinde gerçek bellek alanının yeni boyuta (realloc) uydurulması gerekiyor.

4.3 Sorun: Anahtar kelimelerin hatalı dönüşümü

Bu son sorun genellikle, Türkçe karakter dönüşümlerini doğru olarak yapabilen yeni/modern uygulamalarda ortaya çıkıyor. Uygulama imap, quit gibi anahtar kelimelerini (keyword) Türkçe yerelinde İMAP, QUİT karşılıklarına çeviriyor fakat dönüş değeri olarak İngilizce çevrimleri IMAP ve QUIT'i bekliyor. Sonuç olarak anahtar kelimelerin karşılaştırmaları hatalı sonuç veriyor.

Çözüm olarak yalnızca anahtar kelimelerin dönüşümünde yerelin setlocale() çağrısı ile C yereline çevrilmesi işe yarıyor.

Özellikle bu son sorun için http://www.i18nguy.com/unicode/turkish-i18n.html adresindeki belge gerekli açıklamayı yapıyor.