Pitfall Of Creating An Internationalized Program:

Using setlocale

Written by George Rhoten, IBM Software Developer

When internationalizing a program so that it can work in multiple locales, one of the functions you may consider using is setlocale. This function is a standard part of the C programming language, but it is also available in other programming languages, like PHP. A programmer would want to use this function to change how numbers are formatted, how dates are formatted, which translated text to use and so forth.

Here is an example of how setlocale could be used:

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

int main(int argc, const char* const argv[]) {
   printf("The original locale is %s\n", setlocale(LC_ALL, NULL));
   printf("An example number in this locale is %g\n", 1234.567);
   printf("The default locale is %s\n", setlocale(LC_ALL, ""));
   printf("An example number in this locale is %g\n", 1234.567);
   printf("The default locale is %s\n", setlocale(LC_ALL, "de_DE"));
   printf("An example number in this locale is %g\n", 1234.567);
   return 0;
}
```

The sample output of this program on a Linux machine would be the following:

```
The original locale is C
An example number in this locale is 1234.57
The default locale is en_US.UTF-8
An example number in this locale is 1234.57
The default locale is de_DE
An example number in this locale is 1234,57
```

In this sample, the first call to setlocale queries the currently used locale. By default, this is usually set to the “C” locale, which has a lot of formatting conventions that you would see in the en_US locale (English as used in the United States). The second call to setlocale, changes the default locale to the default locale settings specified by a system. In this case, my Linux machine is set to the en_US locale with the UTF-8 charset. In the third call to setlocale, the default locale of the running program is changed to de_DE (German as used in Germany). The important thing to note is that the decimal point changed from a period to a comma when setlocale was used.

What is wrong?

Now that we have looked at how setlocale can be used to support multiple locales, let us look at some

of the problems with the function. If your single threaded program only needs to work on one operating system and only work under a single locale, there is no problem. Unfortunately, this would also mean that your application isn't portable, scalable or internationalized enough to work for your customers.

Portability

If you read the definition of setlocale function http://www.opengroup.org/onlinepubs/009695399/functions/setlocale.html, you will notice that the locale argument is implementation specific. While the format of the locale identifier is fairly uniform between platforms, the contents of the identifier frequently varies between platforms. Here are some examples of the locales that can be used with setlocale:

Locale	***Windows***	***Linux***	***Solaris***
Chinese (China)	Chinese_People's Republic of China.936	zh_CN.gbk	zh_CN.GBK
Korean (South Korea)	Korean_Korea.949	ko_KR.euckr	ko_KR.EUC
English (United States)	English_United States.1252	en_US.UTF-8	en_US.UTF-8

The first example shows Chinese in China using the GBK codepage. All three platforms represent the same locale in the same codepage. The main difference is how the locale is identified. Linux and Solaris use almost the same identifier, but Windows uses a totally different locale identifier. There is no compatibility with the locale identification.

The second example shows Korean in South Korea. All three operating systems in this example use the EUC-KR codepage. Unfortunately, you will notice that Linux and Solaris use different locale identifiers to represent the same locale information. So the difference in locale identifiers is not just a POSIX versus Windows issue. Hard coding the locale for setlocale into your program is not really a portable solution.

The third example shows English in the United States. In the MSDN documentation from Microsoft, setlocale is documented to not to support UTF-8. If you look at many POSIX based operating systems, like Linux or Solaris, windows-1252 is not a supported codepage for this locale. So if you have translated your API or your user interface to use a specific codepage for a locale, you can not easily store or transfer text data between machines in a certain locales, and you can not expect the same bytes to be interpreted the exact same way without understanding that there is a codepage difference. Many programs get around this problem of different codepages supported an operating system by converting this data to Unicode before any interpretation and converting from Unicode when a non-Unicode capable API must be used. Java, ICU and the Microsoft .NET framework use a similar model where all strings use UTF-16.

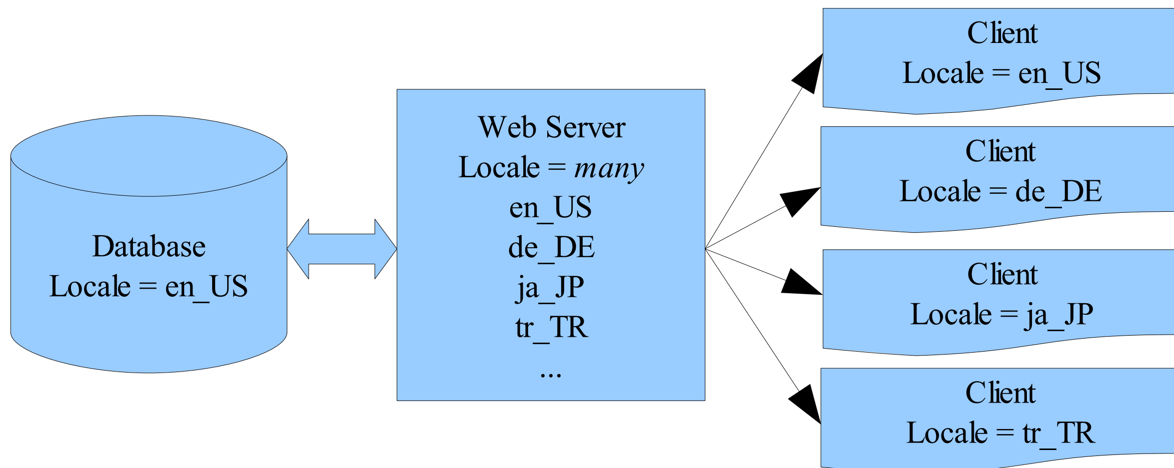
Multithreading Issues

Due to the rise in popularity of multithreaded applications, it is worth noting the scalability of using setlocale. While multithreading an application on single core CPUs would only increase the responsiveness of a program, multicore CPUs increase the performance of your program by executing your code in a truly parallel manner. Unfortunately, setlocale is designed to modify the default locale

of the process, not for the thread. There are some cases where this is undesirable.

If you want to format a number, normalize text via uppercasing or lowercasing, format a date, perform text resource lookup or some other localized operation, you can not perform these multilingual operations with a single global locale. The reason is that you can not modify the default locale by using setlocale unless you use a long mutex lock around all of these types of operations. The C functions to do these operations do not take a locale parameter, and they only take the locale value from setlocale. So the default locale can not change while doing these localized operations. The users of your program probably do not want to see results formatted in one locale, and then have the rest of the result formatted in a different locale. You might see this type of scenario in a client/server environment, where each localized client has work done in one or more threads on the server. You could have the localized work done in multiple processes instead of multiple threads, but threads are much lighter weight than processes and thus scale better in a server environment.

Here is a more visual example that demonstrates this scenario:



In this example, the web server must take data formatted for one locale and display it in a appropriate manner for a specific client. If this setup was a website running an on-line store, it would be best to have the prices and descriptions of the available products displayed differently for each locale. A German client in Germany would typically like to have the descriptions written in German with prices in Euros, and an English client in the United States would typically like to have the descriptions written in English with US dollar prices. Having the default locale format changed in the middle of formatting a product price would be very bad, which could cause the price to appear with the wrong currency symbol.

Not all web sites are setup in this multithreaded manner, but there are other types of applications where this model is frequently needed. A single thread may be formatting a view of some data for a specific locale. The ability to specify the locale for the current operation can be very helpful in these situations.

Alternatives to setlocale and Other C Based Formatting Functions

There are a few alternatives to setlocale. You could rewrite your application into Java. Java allows each formatting operation to specify a specific locale. Microsoft's .NET framework has a similar ability. Unfortunately, completely rewriting an application into a different language or using .NET is

not always a viable solution, since that would involve rewriting a lot of code, and the rewriting process could introduce new bugs into the application. Rewriting code into a different language is not always that simple.

If you plan on keeping your code portable and written in C or C++, you can use ICU (International Components for Unicode). ICU is framework for internationalizing an application.

ICU, .NET and Java have the ability to specify the locale for the formatting operation. In .NET speak, the term culture information is used instead of locale.

Here is one modified example of the first example of this article rewritten to use ICU instead of setlocale. This example uses the C API of the ICU I/O library.

```
#include "unicode/ustdio.h"

int main(int argc, const char* const argv[]) {
   UFILE *stdOutFile = u_finit(stdout, NULL, NULL);
   u_fprintf(stdOutFile, "The original locale is %s\n", u_fgetlocale(stdOutFile));
   u_fprintf(stdOutFile, "An example number in this locale is %g\n", 1234.567);
   u_fsetlocale(stdOutFile, "en_US");
   u_fprintf(stdOutFile, "The default locale is %s\n", u_fgetlocale(stdOutFile));
   u_fprintf(stdOutFile, "An example number in this locale is %g\n", 1234.567);
   u_fsetlocale(stdOutFile, "de_DE");
   u_fprintf(stdOutFile, "The default locale is %s\n", u_fgetlocale(stdOutFile));
   u_fprintf(stdOutFile, "An example number in this locale is %g\n", 1234.567);
   u_fclose(stdOutFile);
   return 0;
}
```

Here are the results from the rewritten example.

```
The original locale is en_US
An example number in this locale is 1,234.57
The default locale is en_US
An example number in this locale is 1,234.57
The default locale is de_DE
An example number in this locale is 1.234,57
```

References

http://www.icu-project.org/ ICU web site
http://www.opengroup.org/onlinepubs/009695399/functions/setlocale.html The POSIX definition of setlocale
http://www.php.net/setlocale The PHP definition of setlocale
http://msdn2.microsoft.com/en-us/library/system.globalization.cultureinfo.aspx .NET information on CultureInfo
http://msdn.microsoft.com/library/en-us/vccore98/HTML/_crt_setlocale.2c_._wsetlocale.asp setlocale information