Bug 23421 - Strange collation rules for A and space with UTF-8 locale when other characters appended

Status:	UNCONFIRMED	**Reported:**	2018-07-17 09:09 IST by Benjamin Cama
Alias:	None	**Modified:**	2018-07-18 08:10 IST (History)
Product:	glibc	**CC List:**	2 users (show)
Component:	localedata (show other bugs)	**See Also:**	
Version:	2.28	**Host:**	
Importance:	P2 normal	**Target:**	
Target Milestone:	---	**Build:**	
Assignee:	Not yet assigned to anyone	**Last reconfirmed:**	
URL:			
Keywords:			
Depends on:			
Blocks:			

Attachments

A and space collation test case (170 bytes, text/x-csrc) 2018-07-17 09:09 IST, Benjamin Cama	Details
Test case for collation between letter and space with later letter appended (170 bytes, text/x-csrc) 2018-07-17 09:22 IST, Benjamin Cama	Details
Add an attachment (proposed patch, testcase, etc.)	View All

Note

You need to log in before you can comment on or make changes to this bug.

Benjamin Cama **2018-07-17 09:09:38 IST** — Description

Created attachment 11136 [details]
A and space collation test case

Hi,
I stumbled against a strange string ordering bug, and managed to reduce it to the attached test case. I *think* it comes from the locale data only, as I can change the difference values slightly (but not the ordering) compared to an older localedata, while still having the exact same behavior between 2.24 libc and latest master (as of two days ago).

Here is the output with git master:

```
% ./testrun.sh ../collate_a_space
setlocale(LC_COLLATE,"C") = 361280657
strcoll("A", " ") = 33
strcoll("AB", " B") = 33
strcoll("B", " ") = 34
strcoll("BB", " B") = 34
setlocale(LC_COLLATE,"en_US.UTF-8") = 380448880
strcoll("A", " ") = 1
strcoll("AB", " B") = -13
strcoll("B", " ") = 1
strcoll("BB", " B") = 1

(the result is exactly the same when not using testrun.sh and only setting LOCPATH)

And with my stock libc (2.24):

% ../collate_a_space
setlocale(LC_COLLATE,"C") = 1774630437
strcoll("A", " ") = 33
strcoll("AB", " B") = 33
strcoll("B", " ") = 34
strcoll("BB", " B") = 34
setlocale(LC_COLLATE,"en_US.UTF-8") = 1082470992
strcoll("A", " ") = 1
strcoll("AB", " B") = -1
strcoll("B", " ") = 1
strcoll("BB", " B") = 1

Note the second strcoll test, which gives an opposite result with an UTF-8 locale compared to raw C one. This only happen with letter “A” (or even “a”), but no other one, hence the “B” test for comparison. And the ordering is correct when comparing lone “A” (or any letter) and “ ” (space), with nothing appended.

Note that this is with 100% ASCII characters.
```

Benjamin Cama **2018-07-17 09:22:59 IST** — Comment 1

Created attachment 11137 [details]
Test case for collation between letter and space with later letter appended

OK, this is not actually about letter “A” only: I reproduce it with any appended letter which is *after* the one we test against space; see attached test case. The different ordering does not happen with a letter *prior* to it appended (replace the “D” with “A” in my example).

Carlos O'Donell **2018-07-17 15:20:57 IST** — Comment 2

(In reply to Benjamin Cama from comment #1)
> Created attachment 11137 [details]
> Test case for collation between letter and space with later letter appended
>
> OK, this is not actually about letter “A” only: I reproduce it with any
> appended letter which is *after* the one we test against space; see attached
> test case. The different ordering does not happen with a letter *prior* to
> it appended (replace the “D” with “A” in my example).

This is expected.

In en_US.UTF-8 the space (as are many special symbols) is ignored for collation.

Therefore "A" < "B" < " B" < " B" < "   B" etc.

Notes:
- On master for localedata/locales/iso14651_t1_common we have:
54827 order_start <SPECIAL>;forward;backward;forward;forward,position
55297 <U0020> IGNORE;IGNORE;IGNORE;<U0020> % SPACE
64325 order_start <LATIN>;forward;backward;forward;forward,position
64347 <U0041> <S0061>;<BASE>;<CAP>;<U0041> % LATIN CAPITAL LETTER A
- This follows the POSIX locale specification:
http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap07.html
"
~~~
The special keyword IGNORE as a weight shall indicate that when strings are compared using the weights at the level where IGNORE is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to IGNORE in their primary weight form an equivalence class.
~~~
- Sorting SPECIAL and LATIN have the same rules, and when you compare "A" to " " it is ignored until the 4th weight where it's compared by Unicode code point, and so results in "A" > " " (which is true.
- Sorting then "A" to " B" skips " " (because IGNORE) and compares "A" to "B" which results in "A" < " B".

If you want full code point sorting you need to use C.UTF-8 which some distributions provide, and which is still not available in upstream glibc (though I'm working on it slowly).

Benjamin Cama **2018-07-17 16:42:30 IST** — Comment 3

Thanks a lot for these explanations! I did not expect that.

My use case is just basic ASCII sorting of space separated values lines using “sort”, which does not work with “default” locales and is quite a PITA. E.g with my default UTF-8 locale (fr_FR.UTF-8):

```
% printf "a b\naa b\n" | sort
aa b
a b
% printf "a b\naa b\n" | LANG=C sort
a b
aa b

This is very weird.

Should I assume that “basic” sorting in Unix should always explicitly state to use the C locale?

Thanks for your time.
```

Carlos O'Donell **2018-07-17 16:54:35 IST** — Comment 4

(In reply to Benjamin Cama from comment #3)
> My use case is just basic ASCII sorting of space separated values lines
> using “sort”, which does not work with “default” locales and is quite a
> PITA. E.g with my default UTF-8 locale (fr_FR.UTF-8):

There is no such thing as "basic ASCII."

You have the C or POSIX locale.

You have a specific locale that you are using.

```
> % printf "a b\naa b\n" | sort
> aa b
> a b

This is correct.

> % printf "a b\naa b\n" | LANG=C sort
> a b
> aa b
>
> This is very weird.
```

Weird is relative ;-)

```
> Should I assume that “basic” sorting in Unix should always explicitly state
> to use the C locale?
```

Please don't call it "basic."

If you want sorting following the C/POSIX rules then use that locale.

If you want code-point sorting for UTF-8 use C.UTF-8 if your distribution offers it.

We just fixed a related issue with ldconfig and sorting of *.conf:
~~~
commit 7d38eb38977980efe703eac93645b1af5a5f8a0c
Author: Aurelien Jarno <aurelien@aurel32.net>
Date:   Sat Dec 16 12:25:41 2017 +0100

    ldconfig: set LC_COLLATE to C [BZ #22505]

    ldconfig supports `include' directives and use the glob function to
    process them. The glob function sort entries according to the LC_COLLATE
    category. When using a standard "include /etc/ld.so.conf.d/*.conf" entry
    in /etc/ld.so.conf, the order therefore depends on the locale used to
    run ldconfig. A few examples of locale specific order that might be
    disturbing in that context compared to the C locale:
    - The cs_CZ and sk_SK locales sort the digits after the letters.
    - The et_EE locale sorts the 'z' between 's' and 't'.

    This patch fixes that by setting LC_COLLATE to C in order to process
    files in deterministic order, independently of the locale used to launch
    ldconfig.

    NOTE: This should NOT be backported to older release branches.

    Changelog:
            [BZ #22505]
            * elf/ldconfig.c (main): Call setlocale to force LC_COLLATE to C.
~~~
Switching to the C/POSIX locale makes this deterministic.

Benjamin Cama **2018-07-18 08:10:40 IST** — Comment 5

Thanks again for the clarification. I understand that this is a POSIX-defined behavior, and I cannot do much about it. Thanks for the example describing a situation where using the C locale is mandated.

I know I cannot convince anyone of changing POSIX, but one last *real* example of “weird” sorting:

```
ENCR_DES        DES_CBC
ENCR_DES_ECB    DES_ECB
ENCR_DES_IV32   DES_IV32
ENCR_DES_IV64   DES_IV64
ENCR_IDEA       IDEA_CBC
ENCR_NULL_AUTH_AES_GMAC NULL_AES_GMAC
ENCR_NULL       NULL

The “usual” rule (as in “historically in Unix, which for a long time used the C/POSIX locale everywhere”; I am speaking of 2000's kind of old, not the 80's, but I am old enough to have lived the Unicode transition in Debian) of having shorter strings sorted before longer ones does not stand (i.e. ENCR_NULL* looks sorted the opposite way of ENCR_DES*). This is with tabs instead of spaces (which have the same ordering rule, it seems), so it stands out more.

It is even stranger in this made up example:

% printf "A\tA\nAA\tA\nA\tD\n"|sort
A       A
AA      A
A       D

I will from now on try not to forget setting the right collation rule before expecting the C sorting behavior. I hope not to be bitten again.

Sorry for the noise and thanks again.
```