uClibc today: Still makes sense

Alexey Brodkin

Embedded Linux Conference Europe 2017

Agenda

What is uClibc

Historical overview

Current state

Comparison to other libc’s

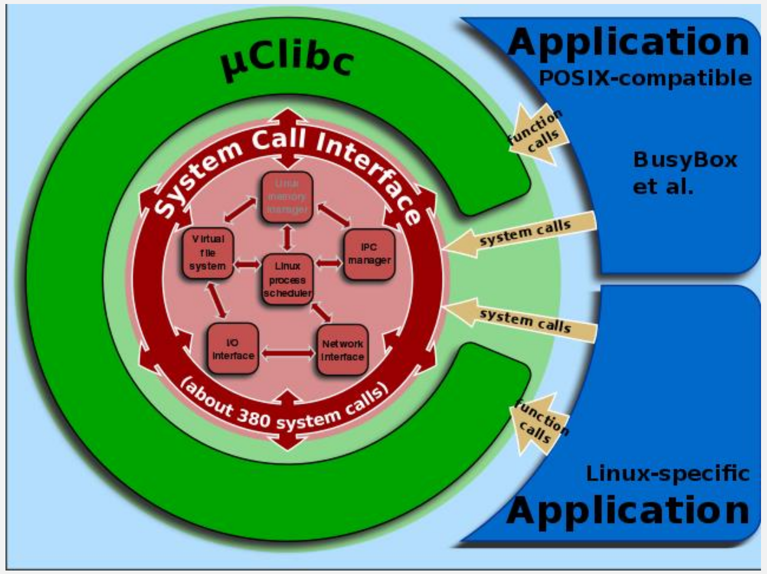
Real life with uClibc

What’s on the roadmap

About the author

Alexey Brodkin

- Sr Software Engineer at Synopsys
- Live and work in St.-Petersburg, Russia
- Maintainer of
  - Linux BSP for Synopsys ARC development systems
  - U-Boot bootloader for ARC architecture
  - Co-maintainer of uClibc for ARC architecture
- Active contributor to
  - Buildroot
  - OpenWrt/Lede



```
static unsigned int
res_randomid(void)
{
    return 0xffff & getpid();
}

int
res_init(void)
{
    …
    _res.id = res_randomid();
```

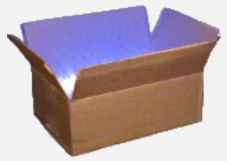
What is uClibc

Compact C library for use with Linux kernel

- C library provides user-space applications interface to Linux kernel via set of standard functions and wrapped syscalls
- Not all libc functions end-up using kernel syscalls:
  - String routines: memcpy(), memmove(), strcmp() etc
  - inet_pton() etc
- Some functions may use a plenty of syscalls:
  - res_init():

```
getpid()
fstatat64(AT_FDCWD, "/etc/resolv.conf"...) = 0
openat(AT_FDCWD, "/ etc/resolv.conf", O_RDONLY) = 3
read(3, ""..., 4096)                        = 103
read(3, "", 4096)                           = 0
close(3)                                    = 0
```

- Syscalls are architecture-dependent thus porting is needed



Busybox web-site:
https://busybox.net/

Busybox git repository:
git://git.busybox.net/busybox

Buildroot web-site:
https://www.buildroot.org/
Buildroot git repository:
git://git.busybox.net/buildroot

Lede Project web-sire:
https://lede-project.org/
Lede Project git repository:
https://git.lede-project.org/?p=source.git

https://git.lede-project.org/?p=openwrt/source.git;a=blob_plain;f=obsolete-buildroot/README;hb=76d90c2ed2

```
-------------->8-----------
This is a modified uClibc buildroot,
customized to build OpenWRT.
-------------->8-----------
```

uClibc++ web-site:
https://cxx.uclibc.org/

uClibc++ git repository:
https://git.busybox.net/uClibc++

Fun facts around uClibc

Did you know?

- Busybox was started before uClibc
- Buildroot was initially created as a testbed for uClibc
- Buildroot is much more popular than uClibc today (judging by activity of developers and casual contributors)
- OpenWrt/Lede uses heavily modified Buildroot as its build system
- There's uClibc++ written by Garrett Kajmowicz which is still used in OpenWrt/Lede by default with Musl & uClibc

```
http://lists.busybox.net/pipermail/uclibc/2000-June/020845.html
> What are the goals of uC-libc ?

To be the smallest fully functional C
library for Linux.

 -Erik
```

```
https://git.uclibc.org/uClibc/commit/?h=64bc6412188b141c010ac3b8e813b837dd991e
commit
64bc6412188b141c010ac3b8e813b837dd991e80
Author: Erik Andersen
<andersen@codepoet.org>
Date:   Sun May 14 04:16:35 2000
+0000

    Initial revision
```

Historical overview

Olde good uClibc: v0.9.1..v0.9.33

- Erik Andersen started development of uClibc in 2000 with x86 and ARM port
- Bernhard Reutner-Fischer became new maintainer in Oct 2008 http://lists.uclibc.org/pipermail/uclibc/2008-October/041191.html
- v0.9.33.2 cut in May 2012
- In the end (in master branch) we had support of:
  - 28 architectures (10 with NPTL)
  - Little- and big-endian machines
  - Shared and static libraries
  - Locales
  - IPv6

Announce:
https://lists.openwrt.org/pipermail/openwrt-devel/2014-July/026922.html

Official web-site:
https://www.uclibc-ng.org/

Main git repository:
https://cgit.openadk.org/cgi/cgit/uclibc-ng.git/

Git repo mirrors:
http://repo.or.cz/w/uclibc-ng.git
https://github.com/wbx-github/uclibc-ng

Mailing list:
https://mailman.uclibc-ng.org/cgi-bin/mailman/listinfo/devel/

Patchwork:
https://patchwork.ozlabs.org/project/uclibc-ng/list/

Historical overview (cont’d)

uClibc-ng: v1.0.0..HEAD

- Waldemar Brodkorb volunteered to create & maintain a fork
- The first release in almost 3 years (v1.0.0 in 2015-02-02)
- Regular releases available at: https://downloads.uclibc-ng.org/releases/
- Run-time regression testing for each release starting from v1.0.5 with results published at: https://tests.embedded-test.org/uClibc-ng/

Announce:
https://lists.openwrt.org/pipermail/openwrt-devel/2014-July/026922.html

Official web-site:
https://www.uclibc-ng.org/

Main git repository:
https://cgit.openadk.org/cgi/cgit/uclibc-ng.git/

Git repo mirrors:
http://repo.or.cz/w/uclibc-ng.git
https://github.com/wbx-github/uclibc-ng

Mailing list:
https://mailman.uclibc-ng.org/cgi-bin/mailman/listinfo/devel/

Patchwork:
https://patchwork.ozlabs.org/project/uclibc-ng/list/

Historical overview (cont'd)

uClibc-ng: v1.0.0..HEAD (cont'd)

Significant changes compared to original master branch:

- Clean-up
  - Removed **e1**, **i960**, **nios**, **sh64**, **v850** and **vax** architectures
  - Removed many configurable options
  - Single libc and de-duplicated threading code
- ABI changes
  - libXXX.so.0 ⇒ libXXX.so.1 (in v1.0.0)
  - libXXX, libYYY, libZZZ ⇒ libc (in v1.0.18)
- New architectures supported
  - **aarch64**, **lm32**, **nds32**, **or1k**, **sparc64**
  - NPTL support for Microblaze & Xtensa
- Separated test-suite with new shell wrapper to execute and generate report (support for noMMU targets)
- More glibc-compatible [by default]
  - malloc(0) returns valid pointer ☺

Current state

Who Uses uClibc today

- Default libc in Buildroot (except PowerPC64 and Sparc64)
- Lilblue Gentoo
  https://wiki.gentoo.org/wiki/Project:Hardened_uClibc/Lilblue
  Security-enhanced, fully featured XFCE4 desktop, amd64 Gentoo system, built with uClibc as its C standard library.
- OpenADK (especially for Or1k and noMMU ARM)
  https://openadk.org/
  Open Source Appliance Development Kit
- OpenWrt/Lede for ARC
- Arches with no other libc's for everything:
  - NDS32
  - Xtensa etc.

Current state (cont’d)

Who no Longer Uses uClibc

- Alpine Linux since June 2014 (v3.0.0), switched to musl https://alpinelinux.org/posts/Alpine-3.0.0-released.html
- OpenWrt/Lede since June 2015 (except for ARC), switched to musl https://lists.openwrt.org/pipermail/openwrt-devel/2015-June/033702.html
- OpenEmbedded since July 2016 (now only glibc & musl) http://git.openembedded.org/openembedded-core/commit/meta/conf/distro?id=ff1599149942af1c36280abd4f1ed3878aaa62eb

Comparison to other libc’s

Most common libc’s used with Linux kernel

- glibc – de-facto standard especially in desktop & server distributions
- uClibc – used to be de-facto standard for embedded Linux
- musl – written from scratch C standard library that is now considered as a uClibc replacement in embedded [and not only embedded] world

Interesting links:

- http://www.etalabs.net/compare_libcs.html
  Detailed comparison of libc’s, still pretty much up-to-date with minor corrections
- http://events.linuxfoundation.org/sites/events/files/slides/libc-talk.pdf
  ELCE2014 presentation gives some criteria for selecting a C library

Comparison to other libc's (cont'd)

Key factors: supported architectures & memory footprint

- Supported architectures:
  - uClibc (**28**): `aarch64, alpha, arc, arm, avr32, bfin, c6x, cris, frv, h8300, hppa, i386, ia64, lm32, m68k, metag, microblaze, mips, mips64, nds32, nios2, or1k, powerpc, sh, sparc, sparc64, x86_64, xtensa`
  - glibc (**18**): `aarch64, arc*, alpha, arm, hppa, i386, ia64, m68k, microblaze, mips, mips64, nios2, powerpc, s390, sh, sparc, tile, x86_64`
  - musl (**12**): `aarch64, arm, i386, microblaze, mips, mips64, or1k, powerpc, powerpc64, s390x, sh, x86_64`

- Sizes (for ARM):
  - uClibc (default): **560 kB**
  - uClibc (-threading, -networking): **330 kB**
  - musl: 600 kB
  - glibc: **2655 kB**

* ARC port is being reviewed now on the mailing list

Real life with uClibc

uClibc is not backward-compatible

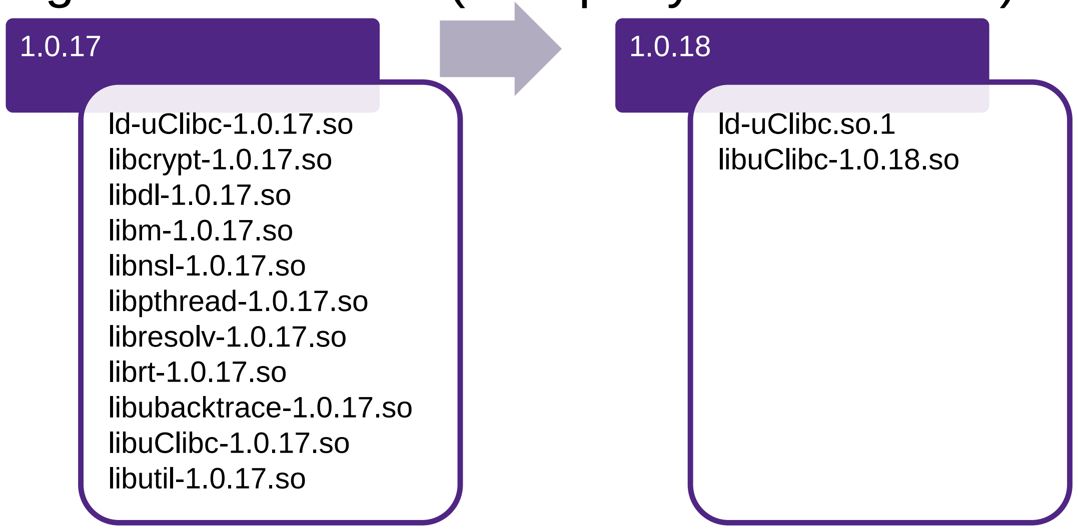
- uClibc-ng bumped version from 0.9.x.y to 1.x.y changing library names [suffixes]
  - GCC still expects ld-uClibc.so.0 (gcc/config/linux.h):

    ```
    #define UCLIBC_DYNAMIC_LINKER32 "/lib/ld-uClibc.so.0"
    ```

    so we created a symlink:

    ```
    ld-uClibc.so.0 -> ld-uClibc.so.1
    ```

  - Apps built against old uClibc expect .so.0 libs while we created .so.1,
    so another series of symlinks for extra backward-compatibility
- In 1.0.18 all libs were merged into one libc (except dynamic loader) similarly to musl



1.0.17

ld-uClibc-1.0.17.so
libcrypt-1.0.17.so
libdl-1.0.17.so
libm-1.0.17.so
libnsl-1.0.17.so
libpthread-1.0.17.so
libresolv-1.0.17.so
librt-1.0.17.so
libubacktrace-1.0.17.so
libuClibc-1.0.17.so
libutil-1.0.17.so

1.0.18

ld-uClibc.so.1
libuClibc-1.0.18.so

Real life with uClibc (cont'd)

__GLIBC__ & __GLIBC_MINOR__ macros used for feature check: fix in uClibc

- uClibc pretends to be glibc 2.2 thus have

```
#define        __GLIBC__      2
#define        __GLIBC_MINOR__      2
```

- Still feature set differs a lot: something extra, something missing

https://cgit.uclibc-ng.org/cgi/cgit/uclibc-ng.git/commit/?id=4a05ed87ceb946608100642121c32e642b58cd0d

glibc compat: bump glibc minor version

```
See this discussion:
http://lists.busybox.net/pipermail/buildroot/2015-August/137229.html

Should help to fix compile issues with boost for ARC.
diff --git a/include/features.h b/include/features.h
index dcf1348..f6fbbf4 100644
--- a/include/features.h
+++ b/include/features.h
#define    __GLIBC__    2
-#define        __GLIBC_MINOR__   2
+#define        __GLIBC_MINOR__   10
 #endif
```

https://cgit.uclibc-ng.org/cgi/cgit/uclibc-ng.git/commit/?id=836c1a7baa9421c1222e022cdc263d8c1a5a2b14

Revert "glibc compat: bump glibc minor version"

```
This reverts commit 4a05ed87ceb946608100642121c32e642b58cd0d.

This breaks SSP detection for gcc, which might be problematic
for some projects. Revert it after some discussion with
buildroot and openembedded people.
```

Real life with uClibc (cont’d)

__GLIBC__ & __GLIBC_MINOR__ macros used for feature check (cont’d): fix in sources

- Add checks for __UCLIBC__ in affected sources

  https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0215d59b154ab90c56c4fe49bc1deefe8bca18f1

```
diff --git a/tools/include/linux/string.h b/tools/include/linux/string.h
index b968794..f436d24 100644
--- a/tools/include/linux/string.h
+++ b/tools/include/linux/string.h
@@ -8,7 +8,11 @@ void *memdup(const void *src, size_t len);

 int strtobool(const char *s, bool *res);

-#ifdef __GLIBC__
+/*
+ * glibc based builds needs the extern while uClibc doesn't.
+ * However uClibc headers also define __GLIBC__ hence the hack below
+ */
+#if defined(__GLIBC__) && !defined(__UCLIBC__)
 extern size_t strlcpy(char *dest, const char *src, size_t size);
 #endif
```

Real life with uClibc (cont’d)

Assumptions for features to always exist (IPv6, locales, libnsl etc)

- uClibc might have some features if configured accordingly
- Some features like libnsl and NSS don’t exist in uClibc
- But we may fix it with autotools/cmake/etc tests during configuration or explicit [de]selection of options

https://git.buildroot.net/buildroot/commit/?id=00e98e69b4a0134823bcc4b626eafb16e77ae4b1

```
diff --git a/package/exim/exim.mk b/package/exim/exim.mk
index b852793..8ad0328 100644
--- a/package/exim/exim.mk
+++ b/package/exim/exim.mk
@@ -72,6 +72,14 @@ define EXIM_USE_DEFAULT_CONFIG_FILE_OPENSSL
 endef
 endif

+# only (e)glibc provides libnsl, remove -lnsl for all other toolchains
+# http://bugs.exim.org/show_bug.cgi?id=1564
+ifeq ($(BR2_TOOLCHAIN_USES_GLIBC),)
+define EXIM_REMOVE_LIBNSL_FROM_MAKEFILE
+	$(SED) 's/-lnsl//g' $(@D)/OS/Makefile-Linux
+endef
+endif
+
 define EXIM_CONFIGURE_TOOLCHAIN
 	$(call exim-config-add,CC,$(TARGET_CC))
 	$(call exim-config-add,CFLAGS,$(TARGET_CFLAGS))
```

Real life with uClibc (cont’d)

uClibc doesn’t support versioning of symbols

Make sure symbols versioning is disabled when building for uClibc

https://sourceware.org/git/?p=elfutils.git;a=commit;h=bafacacaf7659a4933604662daba26a480b29a8d

```
--- a/configure.ac
+++ b/configure.ac
+AC_ARG_ENABLE([symbol-versioning],
+AS_HELP_STRING([--disable-symbol-versioning],
+               [Disable symbol versioning in shared objects]))
+AM_CONDITIONAL(SYMBOL_VERSIONING, [test "x$enable_symbol_versioning" != "xno"])
+AS_IF([test "x$enable_symbol_versioning" = "xno"],
+      [AC_MSG_WARN([Disabling symbol versioning breaks ABI compatibility.])])
+
 dnl The directories with content.
 dnl Documentation.
```

https://git.buildroot.net/buildroot/commit/?id=a3f0785396e64b5e2428f860d785f00bbc665d67

```
--- /dev/null
+++ b/package/elfutils/0007-Allow-disabling-symbol-versioning-at-configure-time.patch
...
diff --git a/package/elfutils/elfutils.mk b/package/elfutils/elfutils.mk
index 227dea9..838c3b8 100644
--- a/package/elfutils/elfutils.mk
+++ b/package/elfutils/elfutils.mk
@@ -34,6 +34,7 @@ ELFUTILS_CONF_ENV += \

 ifeq ($(BR2_TOOLCHAIN_USES_UCLIBC),y)
 ELFUTILS_DEPENDENCIES += argp-standalone
+ELFUTILS_CONF_OPTS += --disable-symbol-versioning
 endif
 ifeq ($(BR2_PACKAGE_ZLIB),y)
```

Real life with uClibc (cont'd)

malloc(0) [used to] return NULL

- glibc's **malloc(0)** returns a "valid" pointer to something
- Before v1.0.21 with disabled MALLOC_GLIBC_COMPAT uClibc's **malloc(0)** returned NULL as well as errno set to ENOMEM
- That caused problems in cases like this:

```
if (!malloc(0)) {
        printf("Error!\n");
}
```

- Since v1.0.21 uClibc returns "valid" pointer as well

Real life with uClibc (conclusion)

It’s not [only] uClibc who’s guilty

- What do we have:
  - uClibc is not backward-compatible
  - uClibc doesn’t implement everything other libc’s do
  - uClibc implements some things differently compared to other libc
  - Many application developers rely on feature-set and implementations as in glibc

- So how to live with that?
  - Keep built toolchain, system librariess and applications in sync
    i.e. upgrade binaries simultaneously
  - In applications check libc features with autotools, cmake etc
  - Send emails to uClibc’s mailing list if something goes terribly wrong

What’s on the roadmap

There’re a lot of things to work on

- Reduce compiler warnings and runtime errors running the test suite
- Complete existing architecture support
  (nios2, alpha, sparc64 and others missing ld.so/NPTL support)
- Add new obscure architecture support (c-sky is in works)
- Get rid of the NPTL dependency to dlopen libgcc_s.so
- FDPIC binary format support for ARM/SH2/J2 noMMU boards
- Keep existing support alive

Summary

uClibc still makes sense

- uClibc is mature and pretty complete implementation of a standard C library
- Its predictable release cycle simplifies life for distributions and build-systems
- In some cases there’s no other option
  - No other C libraries for a given architecture (NDS32)
  - No other C libraries for noMMU hardware (BlackFin, ARM, Xtensa, m68k)
- In some cases there’re other options, but still
  - [downconfigured] uClibc might be more efficient solution
  - uClibc might be as good as other available libc’s [so why not? Look at Lilblue Gentoo]
- In some cases uClibc might not be an [easy] option
  - Someone needs to address differences between default [g]libc and others…
    but [usually] that could be fixed [quite easily] given enough desire, patience and time ☺

Thank You