Uc-libc vs klee-uclibc

Code Structure:

1. One folder for each library (with all the source files)
2. One folder for system dependencies
    1. One folder for each architecture

Libraries:

1. inet
2. pwd_grp
3. signal
4. stdio
5. sodlib
6. string
7. termios
8. unistd
9. misc

Hiding:

1. Conditional Hiding:
    1. *Architecture based:*

```
#if defined __USE_BSD || (defined __USE_XOPEN && !defined __USE_UNIX98)
    libc_hidden_proto(strlen)
    libc_hidden_proto(strcpy)
    libc_hidden_proto(uname)
```

    2. *Other Condition* such

```
#ifndef WANT_WIDE
    libc_hidden_proto(strncmp)
```

2. Unconditional Hiding

```
libc_hidden_proto(open)
libc_hidden_proto(read)
libc_hidden_proto(close)
libc_hidden_proto(gettimeofday)
```

What is libc_hidden_*(function):[1][2]

1. *libc_hidden_proto(method)* is used to disable the definition of the method in the Standard C library.[3]

1b. In other words, they tell the compiler to execute the function defined within the same library.[4]

2. *libc_hidden_** macros are used for PLT bypassing within libc.so (and if needed other libraries similarly). For details, please see EndNote[A].[2]

Overriding of certain methods: Klee-uclibc overrides certain methods of the uclibc library. This is done via the attribute_hidden method.

libc_hidden_proto vs attribute_hidden:

1. *libc_hidden_proto* is used for functions while *attribute_hidden* can also be used for attributes/variables.

   *void *__curbrk attribute_hidden = 0;*

2. *libc_hidden_proto* is used to hide the standard implementation (but does keep a copy of the standard implementation) while *attribute_hidden* only keeps the copy of the new implementation during linking. Please note that this is taken from a blog with respect to R but I guess the concept should hold in C/C++ too.[5]

System Dependencies:

Both uclibc and klee-uclibc contains files specific to linux run on different platforms such as x64, arm etc. Each platform folder has the assembly code (as well as the registers).

Klee-uclibc also adds architecture specific features for system calls like abort().

References:

[1] http://stackoverflow.com/a/37293022
[2] http://www.scs.stanford.edu/histar/src/pkg/uclibc/include/libc-symbols.h
[3] https://github.com/nithishr/Klee-uClibcxx/blob/master/klee-uClibcxx/Reports/CodeAnalysis.pdf
[4] https://sourceware.org/ml/libc-alpha/2015-06/msg00036.html
[5] http://comments.gmane.org/gmane.comp.lib.uclibc.general/23292

Endnotes:

[A] First of all, you need to have the function prototyped somewhere, say in foo/foo.h:

```
int foo (int __bar);

If calls to foo within libc.so should always go to foo defined in libc.so,
then in include/foo.h you add:

libc_hidden_proto (foo)

line and after the foo function definition:

int foo (int __bar)
{
  return __bar;
}
libc_hidden_def (foo)

or

int foo (int __bar)
{
  return __bar;
}
libc_hidden_weak (foo)
```

Similarly for global data. If references to foo within libc.so should always go to foo defined in libc.so, then in include/foo.h you add:

```
libc_hidden_proto (foo)

line and after foo's definition:

int foo = INITIAL_FOO_VALUE;
libc_hidden_data_def (foo)

or

int foo = INITIAL_FOO_VALUE;
```

```
  libc_hidden_data_weak (foo)

  If foo is normally just an alias (strong or weak) to some other function,
  you should use the normal strong_alias first, then add libc_hidden_def
  or libc_hidden_weak:

  int baz (int __bar)
  {
    return __bar;
  }
  strong_alias (baz, foo)
  libc_hidden_weak (foo)

  If the function should be internal to multiple objects, say ld.so and
  libc.so, the best way is to use:

  #if !defined NOT_IN_libc || defined IS_IN_rtld
  hidden_proto (foo)
  #endif

in include/foo.h and the normal macros at all function definitions
  depending on what DSO they belong to.

  If versioned_symbol macro is used to define foo, libc_hidden_ver macro should be used, as in:

  int __real_foo (int __bar)
  {
    return __bar;
  }
  versioned_symbol (libc, __real_foo, foo, GLIBC_2_1);
  libc_hidden_ver (__real_foo, foo)  */
```