Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

The Role of C Libraries in a Modern Buildsystem

Alejandro Hernandez Samaniego
Microsoft
*Views are my own

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Outline

- Toolchain components
- C Libraries available
  - Linux
  - Baremetal
- Buildsystem availability
  - Yocto
  - Buildroot
  - Other OSs (Mainstream)
- Case Study: OpenEmbedded + picolibc

Toolchain

Toolchains

- *Distinct software development tools that are linked (chained) together by specific stages.*

- Binutils
  - as
  - ld
- Compiler
  - gcc
  - clang
- C Library
- Debugger
  - gdb
  - lldb

Linux C Libraries

Linux C libraries

C Standard Library / ISO C Library / libc

The C standard library provides macros, type definitions and functions for tasks such as string handling, mathematical computations, input/output processing, memory management, and several other operating system services.

POSIX C Library - Superset of ISO C Library

- glibc
- musl
- uclibc / uclibc-ng
- Bionic – libhybris (compat layer)
- klibc (2023), dietlibc (2018)

glibc - GNU C Library

The project provides the core libraries for the GNU system and GNU/Linux systems, as well as many other systems that use Linux as the kernel. These libraries provide critical APIs including ISO C11, POSIX.1-2008, BSD, OS-specific APIs and more.

- GNU Project
- Designed to be backwards compatible
- **Portable**
- High performance
- Aims to follow all relevant **standard**s including ISO C11, POSIX.1-2008, and IEEE 754-2008
- GPL-2 (or newer) Licensed
- **Compatibility**
- Most widely used
- Less secure?
- Larger footprint
  - Storage
  - RAM

musl

musl is an implementation of the standard library functionality described in the ISO C and POSIX standards, plus common extensions, built on top of the Linux system calls API. musl is lightweight, fast, simple, free, and strives to be correct in the sense of standards-conformance and safety.

- **Simplicity**
- **Code correctness**
- More Secure / Smaller attack surface
- Avoid supporting old architectures - Less compatibility
- Smaller footprint
  - Storage
  - RAM
- Only one lib file (-lm -lpthread not necessary, libm.a is empty)
- Some claims of it being slower for some workloads
  - String operations
  - More I/O operations
  - Slower allocations? - newer malloc implementation
  - No micro-architecture specific optimizations
- MIT License

[1] https://elinux.org/images/e/eb/Transitioning_From_uclibc_to_musl_for_Embedded_Development.pdf (Rich Felker, ELC 2015)

uclibc / uclibc-ng

Development on hold - uclibc-ng

uClibc-ng is a small C library for developing embedded Linux systems. It is much smaller than the GNU C Library, but nearly all applications supported by glibc also work perfectly with uClibc-ng.

uClibc-ng is a spin-off of uClibc (from Erik Andersen) from http://www.uclibc.org. Our main goal is to provide regulary a stable and tested release to make embedded system developers happy.

Originally developed to support uClinux (Linux for MMU-less Microcontrollers)
- glibc lacked support for MMU-less systems
- smaller footprint
- provide as much functionality as possible in a small amount of space
- Targets embedded Linux devices
- full C99 Math library support, wordexp, IPV6, and RPC support disabled by default
- LGPL Licensed
- Supports a myriad of architectures (musl < uclibc < glibc)
- uses glibc headers vs musl uses its own
- Specific purpose vs General purpose (musl and glibc)

Bionic

- Developed my Google
- Specific for Android devices
  - Less memory
  - < Processing power
- Libc + libm + libdl + libpthread(libc)
- BSD license (avoids glibc’s GPL)
- Own’s ABI – cannot be replaced by a different C library
- Smaller than glibc
- Mostly POSIX (deliberately printf not conformant: no %n format)
- May use libhybris as compatibility layer

Baremetal C Libraries

Baremetal C Libraries

- Newlib
- AVR-libc
- Picolibc

newlib

Newlib is a C library intended for use on embedded systems. It is a conglomeration of several library parts, all under free software licenses that make them easily usable on embedded products.

- Embedded specific
- Typically provided by a vendor toolchain e.g. ARM Embedded Toolchain, Renesas
- A few low level routines (glue) needed to port it to a new arch [1]
- Stubs are sometimes required
- BSP separated from main code (libgloss)
- Autotools
- Widely used
- GPL | BSD | Apache | TCL Licensed

[1] https://wiki.osdev.org/Porting_Newlib

AVR-libc

This is the standard library for Microchip (formerly Atmel) AVR devices together with the AVR-GCC compiler. Provides a subset of the standard C library for Atmel AVR 8-bit RISC microcontrollers.

- Most functionality required by ISO standard
- A lot of auxiliary functionality specific for AVR
- Provides basic but customizable startup code
- Works together with Binutils linker scripts
  - Typically no need to provide specifics
- Functions are not guaranteed to be reentrant (undefined behavior if used on interrupt context)
- BSD Licensed

picolibc

- Based on newlib
  - Newlib is too big
  - Requires many support routines via libgloss (e.g. sbrk is missing)
  - stdio uses a lot of code and requires a lot of RAM
  - No integrated test suite
  - No reason to still use libgloss when there is no OS
  - struct_reent usage of RAM
  - Old code base, C has changed a lot since
- Use newlib math, string locale implementations
- stdio adapted from AVR libc
- struct_reent removed, replaces with Thread Local Storage
- sample startup code, linker scripts, specs, mechanism to get error codes (which allows for testing suite to work on Linux host)
- Semihosting support (I/O communication with host/debugger)
- C18 based
- new build system (meson)
- BSD Licensed (all non-BSD code was removed, printf test cases exception)

  ...

[1] Using picolibc in Embedded Systems, Keith Packard, OSS 2024

Buildsystems

Yocto Project / OpenEmbedded

Yocto Project / OpenEmbedded

- Build a custom OS but also capable of creating a cross toolchain

- Available C libraries:
  - glibc
  - musl
  - baremetal
  - newlib
  - picolibc*

Yocto Project / OpenEmbedded

Typically on a DISTRO.conf

```
TCLIBC="musl"
```

```
ls meta/conf/distro/include/tclibc-*
-rw-r--r-- 1 user user 1.4K Aug  8 18:42 conf/distro/include/tclibc-baremetal.inc
-rw-r--r-- 1 user user  894 Aug  8 18:42 conf/distro/include/tclibc-glibc.inc
-rw-r--r-- 1 user user  700 Aug  8 18:42 conf/distro/include/tclibc-musl.inc
-rw-r--r-- 1 user user 1.4K Aug 29 23:11 conf/distro/include/tclibc-newlib.inc
-rw-r--r-- 1 user user 1.1K Aug 29 23:11 conf/distro/include/tclibc-picolibc.inc
```

Yocto Project / OpenEmbedded

tclibc-musl.inc

```
LIBCEXTENSION = "-musl"

LIBCOVERRIDE = ":libc-musl"

PREFERRED_PROVIDER_virtual/libc ?= "musl"
PREFERRED_PROVIDER_virtual/libiconv ?= "musl"
PREFERRED_PROVIDER_virtual/libintl ?= "musl"
PREFERRED_PROVIDER_virtual/crypt ?= "musl"
PREFERRED_PROVIDER_virtual/libc-locale ?= "musl-locales"
PREFERRED_PROVIDER_virtual/nativesdk-libintl ?= "nativesdk-glibc"
PREFERRED_PROVIDER_virtual/nativesdk-libiconv ?= "nativesdk-glibc"

LIBC_DEPENDENCIES = "\
    musl \
    musl-dbg \
    musl-dev \
    musl-utils \
    musl-utils-iconv \
    bsd-headers-dev \
    "
```

Yocto Project / OpenEmbedded

- Compatibility (Embedded) / Patches required:
  - glibc: 23 patches
  - musl: 2 patches
  - Both are updated regularly on stable releases

Yocto Project / OpenEmbedded

Size comparison (master branch 08/2024):

glibc:

```
$ ls -lh tmp/deploy/images/qemuarm64/
-rw-r--r-- 2 user user  21M Aug 30 00:00 Image--6.10.3+git0+5161bedbdc_92466d9d49-r0-qemuarm64-20240830051533.bin
-rw-r--r-- 2 user user  22M Sep  2 00:37 core-image-minimal-qemuarm64.rootfs-20240830051533.ext4
```

musl:

```
$ ls -lh tmp/deploy/images/qemuarm64/
-rw-r--r-- 2 user user  21M Sep  2 00:53 Image--6.10.3+git0+5161bedbdc_92466d9d49-r0-qemuarm64-20240902064008.bin
-rw-r--r-- 2 user user  17M Sep  2 01:12 core-image-minimal-qemuarm64.rootfs-20240902064008.ext4
```

Buildroot

Buildroot

- Available C libraries:
  - glibc
  - musl
  - uclibc-ng
  - newlib (2024 Xilinx Microblaze, ARM64 fails when building libgcc)

- menuconfig
  - Toolchain -> C Library
  - Toolchain -> Build bare metal toolchain

Buildroot

Default build ARM64 + which

- glibc:

```
$ ls -lh output_glibc/images/
total 182M
-rw-r--r-- 1 user user 42M Sep  2 21:15 Image
-rw-r--r-- 1 user user 80M Sep  2 21:15 rootfs.ext2
```

- musl:

```
$ ls -lh output/images/
-rw-r--r-- 1 user user 42M Sep  2 21:51 Image
-rw-r--r-- 1 user user 80M Sep  2 21:51 rootfs.ext2
```

Buildroot

```
$ file glibc/usr/bin/which
glibc/usr/bin/which: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 6.8.0, stripped

$ ls -lh glibc/usr/bin/which
-rwxr-xr-x 1 user user 27K Sep  3 00:47 glibc/usr/bin/which


$ file musl/usr/bin/which
musl/usr/bin/which: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked,
interpreter /lib/ld-musl-aarch64.so.1, stripped

$ ls -lh musl/usr/bin/which
-rwxr-xr-x 1 user user 23K Sep  2 23:42 musl/usr/bin/which
```

Other Linux OS's (conventional)

Ubuntu

- Uses glibc by default
- musl available via wrapper
  - musl-gcc

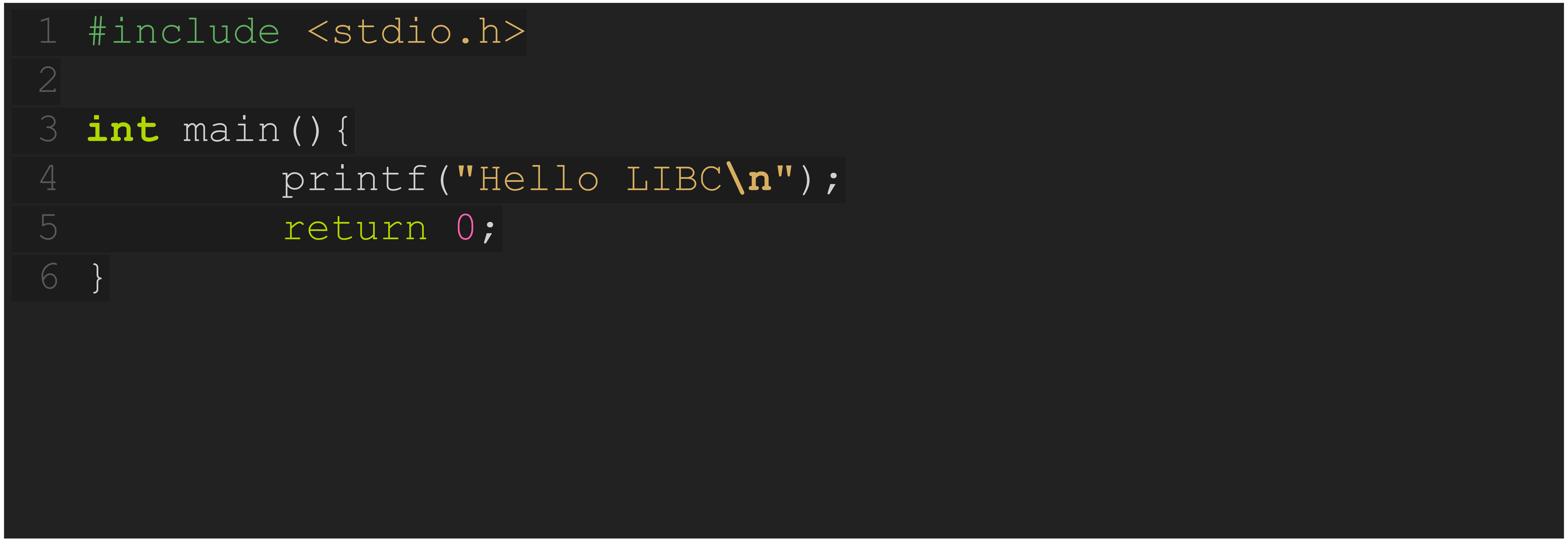
```
$ apt install musl:
/lib/ld-musl-x86_64.so.1
/lib/x86_64-linux-musl/libc.so
```

```
$ apt install musl-tools
$ which musl-gcc
/usr/bin/musl-gcc # same for musl-ldd
```

```
root@131b03420721:/# ls -lh /lib/x86_64-linux-musl/libc.so
-rwxr-xr-x 1 root root 715K Nov 10  2023 /lib/x86_64-linux-musl/libc.so
```

```
root@131b03420721:/# ls -lh /lib/x86_64-linux-gnu/libc.so.6
-rwxr-xr-x 1 root root 2.1M Aug  8 14:47 /lib/x86_64-linux-gnu/libc.so.6
```

Ubuntu



```c
#include <stdio.h>

int main(){
        printf("Hello LIBC\n");
        return 0;
}
```

Ubuntu

```
root@131b03420721:~# gcc libc-test.c -o hello-glibc

root@131b03420721:~# ldd hello-glibc
        linux-vdso.so.1 (0x000074ef261df000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000074ef25e00000)
        /lib64/ld-linux-x86-64.so.2 (0x000074ef261e1000)

root@131b03420721:~# ./hello-glibc
Hello LIBC
```

Ubuntu

```
root@131b03420721:~# musl-gcc libc-test.c -o hello-musl

root@131b03420721:~# ldd hello-musl
./hello-musl: error while loading shared libraries: /lib/x86_64-linux-gnu/libc.so: invalid ELF header

root@131b03420721:~# musl-ldd hello-musl
	/lib/ld-musl-x86_64.so.1 (0x728d4f4c4000)
	libc.so => /lib/ld-musl-x86_64.so.1 (0x728d4f4c4000)

root@131b03420721:~# ./hello-musl
Hello LIBC
```

Fedora

- glibc by default
- musl available via wrappers
  - musl-gcc
  - musl-clang

```
[root@f0c597af3794 /]# ldd hello_glibc
        linux-vdso.so.1 (0x00007084b813f000)
        libc.so.6 => /lib64/libc.so.6 (0x00007084b7f45000)
        /lib64/ld-linux-x86-64.so.2 (0x00007084b8141000)

[root@f0c597af3794 /]# ldd hello_musl
        linux-vdso.so.1 (0x00007847fc448000)
        ld-musl-x86_64.so.1 => /lib/ld-musl-x86_64.so.1 (0x00007847fc370000)
```

Alpine Linux

- General purpose (container oriented)
- Small: busybox + musl
- OpenRC

```
/ # find /lib | grep ld
/lib/ld-musl-x86_64.so.1

/ # ldd /usr/bin/which
        /lib/ld-musl-x86_64.so.1 (0x7de23b417000)
        libc.musl-x86_64.so.1 => /lib/ld-musl-x86_64.so.1 (0x7de23b417000)
```

Void Linux

All compatible packages in our official repositories are available with musl-linked binaries in addition to their glibc counterparts.

- Nvidia doesn't support musl
- glibc chroot to circumvent that
- flatpaks

Which one should be used?

IT DEPENDS

Pros / Cons

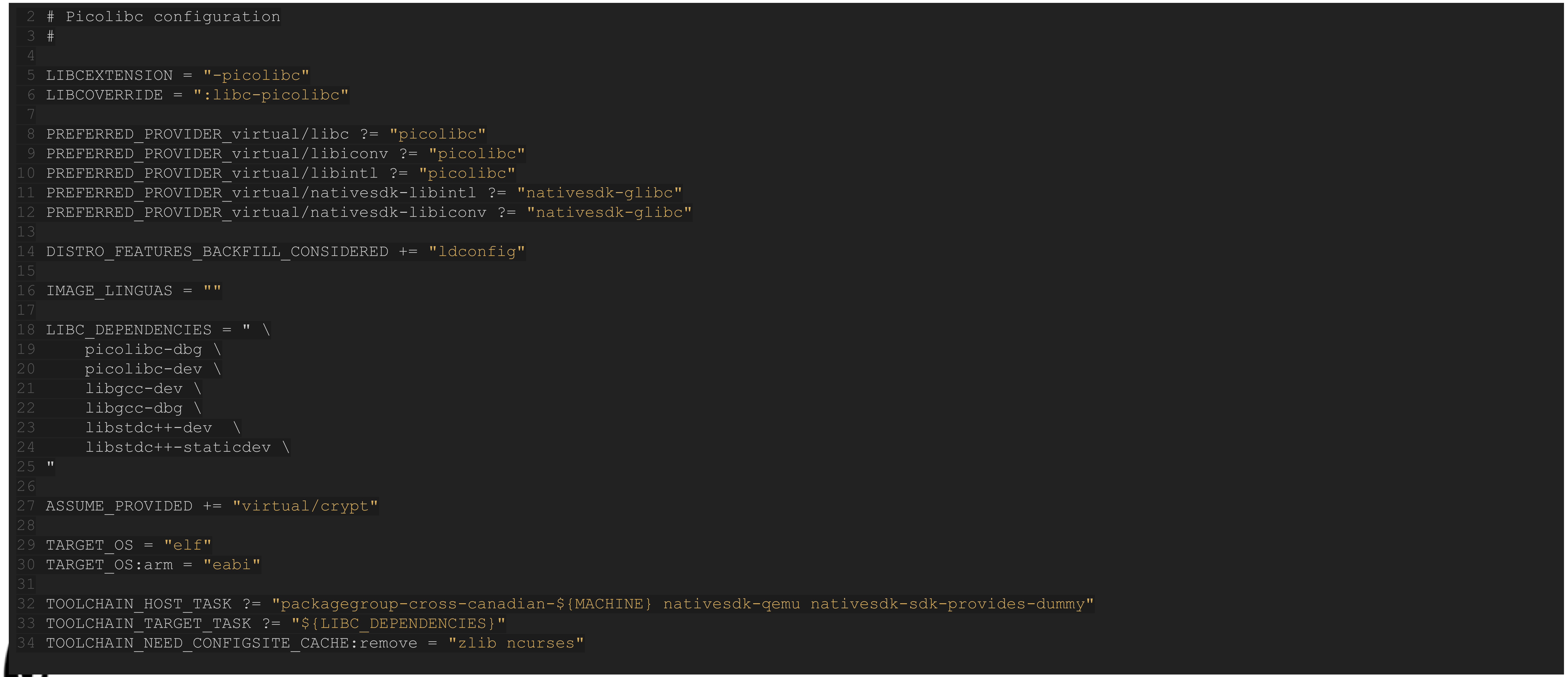
- What is more valuable for your specific need?
  - Size
  - Compatibility
  - Performance
  - Security
  - Flexibility

Case Study: Picolibc on OpenEmbedded

Picolibc case study

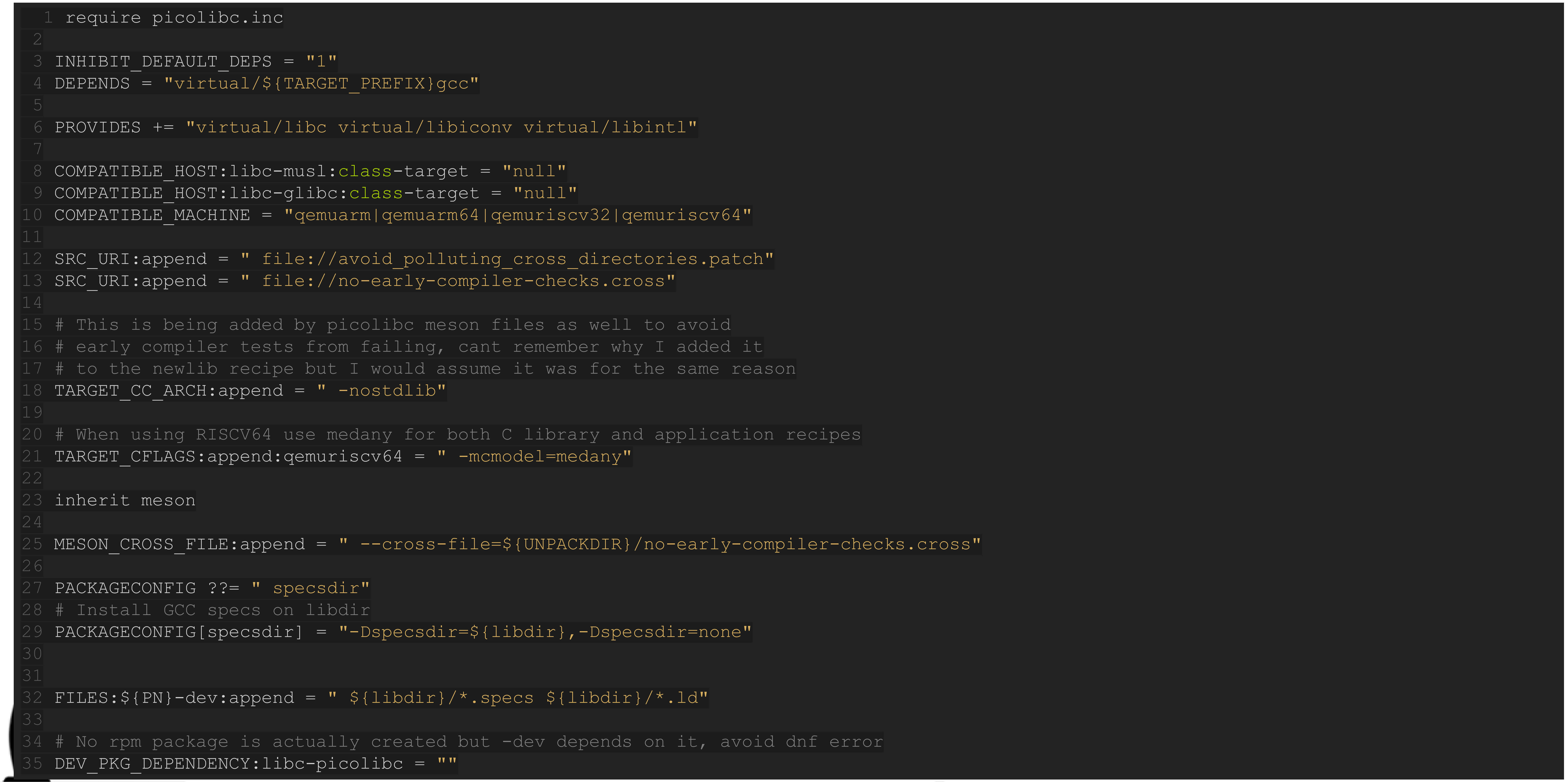
- Adding a new C library to OpenEmbedded
- tclibc-picolibc.inc
- Leverage baremetal-image.bbclass
- Recipe
  - picolibc_git.bb
  - picolibc-helloworld.bb
- Patches to upstream
- selftest/cases/picolibc.py

Picolibc case study



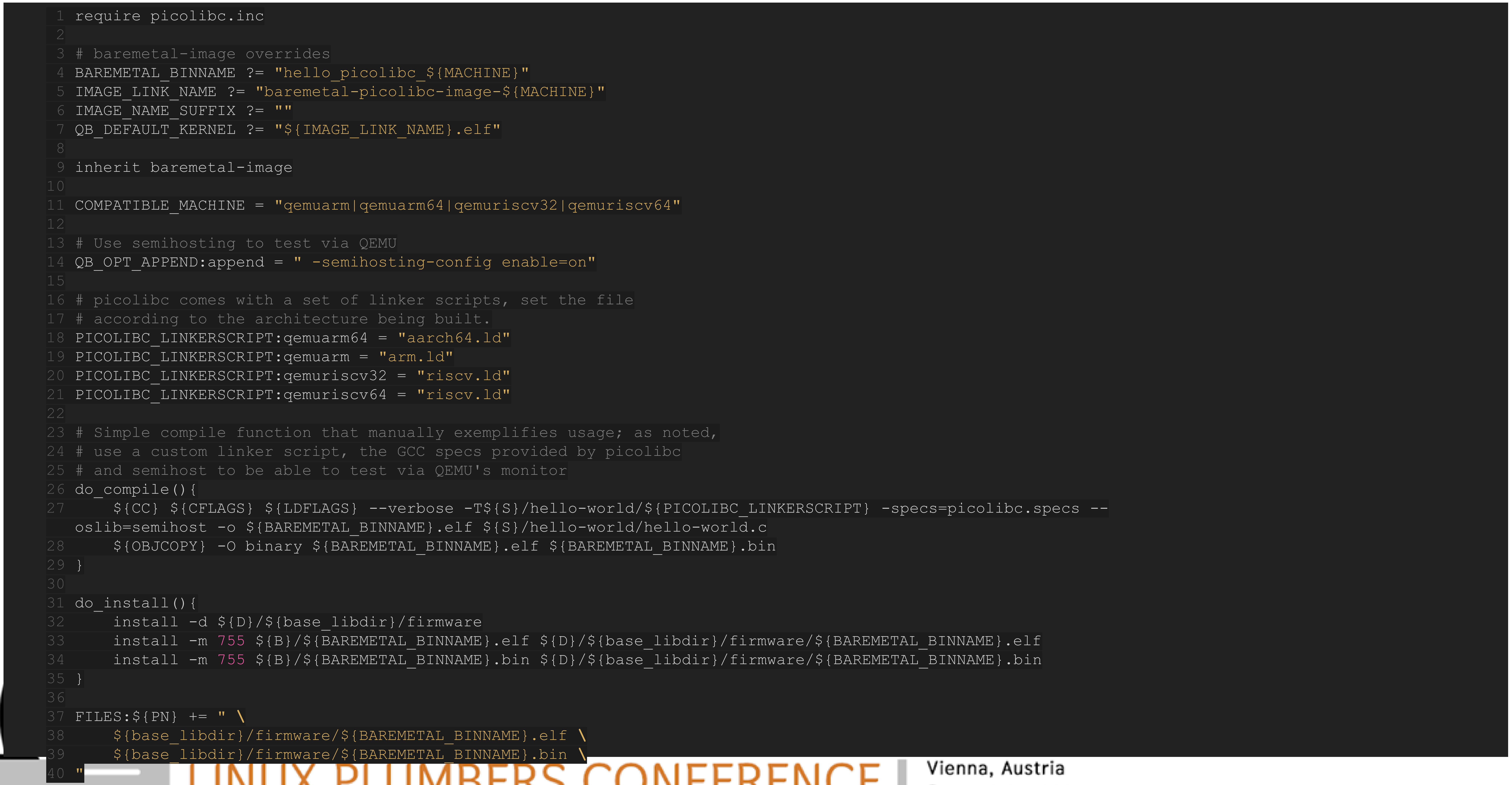
```
# Picolibc configuration
#

LIBCEXTENSION = "-picolibc"
LIBCOVERRIDE = ":libc-picolibc"

PREFERRED_PROVIDER_virtual/libc ?= "picolibc"
PREFERRED_PROVIDER_virtual/libiconv ?= "picolibc"
PREFERRED_PROVIDER_virtual/libintl ?= "picolibc"
PREFERRED_PROVIDER_virtual/nativesdk-libintl ?= "nativesdk-glibc"
PREFERRED_PROVIDER_virtual/nativesdk-libiconv ?= "nativesdk-glibc"

DISTRO_FEATURES_BACKFILL_CONSIDERED += "ldconfig"

IMAGE_LINGUAS = ""

LIBC_DEPENDENCIES = " \
    picolibc-dbg \
    picolibc-dev \
    libgcc-dev \
    libgcc-dbg \
    libstdc++-dev  \
    libstdc++-staticdev \
"

ASSUME_PROVIDED += "virtual/crypt"

TARGET_OS = "elf"
TARGET_OS:arm = "eabi"

TOOLCHAIN_HOST_TASK ?= "packagegroup-cross-canadian-${MACHINE} nativesdk-qemu nativesdk-sdk-provides-dummy"
TOOLCHAIN_TARGET_TASK ?= "${LIBC_DEPENDENCIES}"
TOOLCHAIN_NEED_CONFIGSITE_CACHE:remove = "zlib ncurses"
```

Picolibc case study



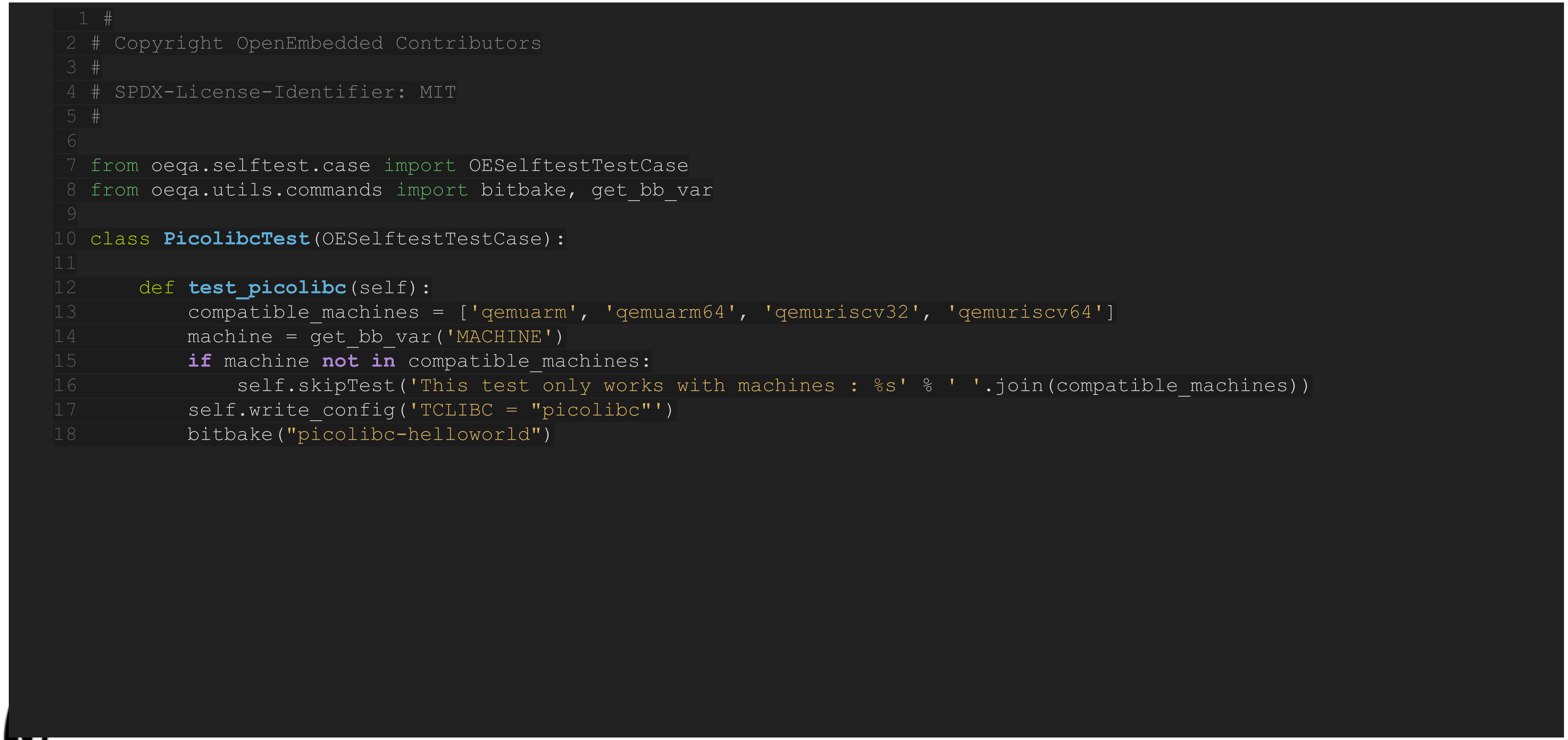
```
require picolibc.inc

INHIBIT_DEFAULT_DEPS = "1"
DEPENDS = "virtual/${TARGET_PREFIX}gcc"

PROVIDES += "virtual/libc virtual/libiconv virtual/libintl"

COMPATIBLE_HOST:libc-musl:class-target = "null"
COMPATIBLE_HOST:libc-glibc:class-target = "null"
COMPATIBLE_MACHINE = "qemuarm|qemuarm64|qemuriscv32|qemuriscv64"

SRC_URI:append = " file://avoid_polluting_cross_directories.patch"
SRC_URI:append = " file://no-early-compiler-checks.cross"

# This is being added by picolibc meson files as well to avoid
# early compiler tests from failing, cant remember why I added it
# to the newlib recipe but I would assume it was for the same reason
TARGET_CC_ARCH:append = " -nostdlib"

# When using RISCV64 use medany for both C library and application recipes
TARGET_CFLAGS:append:qemuriscv64 = " -mcmodel=medany"

inherit meson

MESON_CROSS_FILE:append = " --cross-file=${UNPACKDIR}/no-early-compiler-checks.cross"

PACKAGECONFIG ??= " specsdir"
# Install GCC specs on libdir
PACKAGECONFIG[specsdir] = "-Dspecsdir=${libdir},-Dspecsdir=none"


FILES:${PN}-dev:append = " ${libdir}/*.specs ${libdir}/*.ld"

# No rpm package is actually created but -dev depends on it, avoid dnf error
DEV_PKG_DEPENDENCY:libc-picolibc = ""
```

Picolibc case study



```
require picolibc.inc

# baremetal-image overrides
BAREMETAL_BINNAME ?= "hello_picolibc_${MACHINE}"
IMAGE_LINK_NAME ?= "baremetal-picolibc-image-${MACHINE}"
IMAGE_NAME_SUFFIX ?= ""
QB_DEFAULT_KERNEL ?= "${IMAGE_LINK_NAME}.elf"

inherit baremetal-image

COMPATIBLE_MACHINE = "qemuarm|qemuarm64|qemuriscv32|qemuriscv64"

# Use semihosting to test via QEMU
QB_OPT_APPEND:append = " -semihosting-config enable=on"

# picolibc comes with a set of linker scripts, set the file
# according to the architecture being built.
PICOLIBC_LINKERSCRIPT:qemuarm64 = "aarch64.ld"
PICOLIBC_LINKERSCRIPT:qemuarm = "arm.ld"
PICOLIBC_LINKERSCRIPT:qemuriscv32 = "riscv.ld"
PICOLIBC_LINKERSCRIPT:qemuriscv64 = "riscv.ld"

# Simple compile function that manually exemplifies usage; as noted,
# use a custom linker script, the GCC specs provided by picolibc
# and semihost to be able to test via QEMU's monitor
do_compile(){
    ${CC} ${CFLAGS} ${LDFLAGS} --verbose -T${S}/hello-world/${PICOLIBC_LINKERSCRIPT} -specs=picolibc.specs --oslib=semihost -o ${BAREMETAL_BINNAME}.elf ${S}/hello-world/hello-world.c
    ${OBJCOPY} -O binary ${BAREMETAL_BINNAME}.elf ${BAREMETAL_BINNAME}.bin
}

do_install(){
    install -d ${D}/${base_libdir}/firmware
    install -m 755 ${B}/${BAREMETAL_BINNAME}.elf ${D}/${base_libdir}/firmware/${BAREMETAL_BINNAME}.elf
    install -m 755 ${B}/${BAREMETAL_BINNAME}.bin ${D}/${base_libdir}/firmware/${BAREMETAL_BINNAME}.bin
}

FILES:${PN} += " \
    ${base_libdir}/firmware/${BAREMETAL_BINNAME}.elf \
    ${base_libdir}/firmware/${BAREMETAL_BINNAME}.bin \
"
```

Picolibc case study



```python
#
# Copyright OpenEmbedded Contributors
#
# SPDX-License-Identifier: MIT
#

from oeqa.selftest.case import OESelftestTestCase
from oeqa.utils.commands import bitbake, get_bb_var

class PicolibcTest(OESelftestTestCase):

    def test_picolibc(self):
        compatible_machines = ['qemuarm', 'qemuarm64', 'qemuriscv32', 'qemuriscv64']
        machine = get_bb_var('MACHINE')
        if machine not in compatible_machines:
            self.skipTest('This test only works with machines : %s' % ' '.join(compatible_machines))
        self.write_config('TCLIBC = "picolibc"')
        bitbake("picolibc-helloworld")
```

Picolibc case study

```
$ oe-core/poky/builds/qemuriscv64/tmp/work/x86_64-linux/qemu-helper-native/1.0/recipe-
sysroot-native/usr/bin/qemu-system-riscv64 -device virtio-net-
device,netdev=net0,mac=52:54:00:12:34:02 -netdev
tap,id=net0,ifname=tap0,script=no,downscript=no -object rng-
random,filename=/dev/urandom,id=rng0 -device virtio-rng-pci,rng=rng0 -drive
id=disk0,file=oe-core/poky/builds/qemuriscv64/tmp/deploy/images/qemuriscv64/baremetal-
picolibc-image-qemuriscv64.bin,if=none,format=raw -device virtio-blk-device,drive=disk0
-device qemu-xhci -device usb-tablet -device usb-kbd -nographic -bios none -semihosting-
config enable=on  -machine virt  -m 256 -serial mon:stdio -serial null -nographic -
device bochs-display -kernel oe-
core/poky/builds/qemuriscv64/tmp/deploy/images/qemuriscv64/baremetal-picolibc-image-
qemuriscv64.elf -append 'root=/dev/vda rw  mem=256M
ip=192.168.7.2::192.168.7.1:255.255.255.0::eth0:off:8.8.8.8 net.ifnames=0 console=ttyS0
console=hvc0 earlycon=sbi swiotlb=0'
hello, world


$ qemu-system-riscv64  -bios none -semihosting-config enable=on  -machine virt  -m 256 -
serial mon:stdio -serial null -nographic  -kernel
tmp/deploy/images/qemuriscv64/baremetal-picolibc-image-qemuriscv64.elf
hello, world
```

Summary

- Toolchains
- C Libraries available
- Buildsystems:
  - Availability
  - Usage
  - Comparison
- Case Study: OpenEmbedded + picolibc

Thank you!

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Alejandro Hernandez
alejandro.hernandez@microsoft.com