

2025.7.1-2025.7.7工作记录

工作进展

1. 尝试实现在rt-smart上静态链接rust程序生成的库。只成功兼容了链接工具链，并未实现rust调用rt-smart串口打印函数。
2. 构建一个到 riscv64-unknown-rtsmart 的Rust语言的编译目标，使得Rust编译器可以将程序编译成能够在riscv64 平台上的rt-smart操作系统上运行的指令序列。
3. 参考社区前人完成的 RUST support for rt-thread项目，希望复现APP项目运行，再推广。

资料收集

rust 调用 rt-smart 的 rt_kprintf 函数: [【完整版】使用 Rust 进行嵌入式开发51CTO博客rust嵌入式开发](#), 博客构建的rust程序调用 rt_kprintf 函数, 并生成库文件, 随后静态链接入内核执行。

构建一个到 riscv64-unknown-rtsmart 的Rust语言的编译目标: <https://github.com/diandianjunA/RT-Thread-Smart-Rust>。

RUST support for rt-thread项目: [RT-Thread-Mirror/rtt_rust](#)

搭建开发环境

安装qemu-system-riscv64

目标平台为riscv64, 因此我们首先需要安装qemu-system-riscv64, 用于支持rt-smart内核

```
sudo apt install qemu-system-riscv64
```

安装musl gcc工具链

然后需要安装 musl gcc 工具链, 下载地址为: https://download.rt-thread.org/download/rt-smart/toolchains/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu_latest.tar.bz2

然后配置环境变量:

```
# riscv64 musl gcc

export RTT_CC=gcc
export RTT_EXEC_PATH=/opt/riscv64-linux-musleabi_for_x86_64-pc-linux-gnu/bin
export RTT_CC_PREFIX=riscv64-unknown-linux-musl-
export PATH=/opt/riscv64-linux-musleabi_for_x86_64-pc-linux-gnu/bin:$PATH
```

使用命令 `source ~/.bashrc` 刷新环境变量配置文件

```

$ qemu-system-git(min) X riscv64-unknown-linux-musl gcc -v
Using built-in specs.
COLLECT_GCC=riscv64-unknown-linux-musl-gcc
COLLECT_LTO_WRAPPER=riscv64-unknown-linux-musl/libexec/gcc/riscv64-unknown-linux-musl/10.1.0/lto-wrapper
Target: riscv64-unknown-linux-musl
Configured with: /builds/alliance/risc-v/toolchain/riscv-config --target=riscv64-unknown-linux-musl --prefix=/builds/alliance/risc-v/toolchain/install-native/ --with-syroot=/builds/alliance/risc-v/toolchain/install-native/riscv64-unknown-linux-musl --with-system-lib=enable-shared --enable-languages=c,c++ --disable-lldwrap --disable-lldbp --disable-libquadmath --disable-libsanitizer --disable_nls --disable-bootstrap --src=/builds/alliance/risc-v/toolchain/install-native/ --with-abi=lp64 --with-arch=rv64imc --with-tune=rv64imc --CFLAGS_FOR_TARGET=-O2 --model=medany --arch=rv64imc --abi=lp64 -D_riscv_soft_float --CXXFLAGS_FOR_TARGET=-O2 --model=medany --arch=rv64imc --abi=lp64 -D_riscv_soft_float
Thread model: posix
Supported LTO compression algorithms: zlib
version 10.1.0 (GCC)

```

重点需要注意 `-march=rv64imac`、`-mabi=lp64` 这两个配置。

安装xmake和scons工具

```
sudo add-apt-repository ppa:xmake-io/xmake
sudo apt update
sudo apt install xmake
sudo apt-get install scons
```

安装ncurses库

```
sudo apt-get install libncurses5-dev
```

构建内核镜像

进入到 `machines/qemu-virt-aarch64` 目录下

- ## 1. 选择RT-Thread Kernel选项

```
(Top)
RT-Thread Kernel --->
  AArch64 Architecture Configuration --->
  (0xffff000000000000) The virtual address of kernel start
  RT-Thread Components --->
  RT-Thread Utestcases --->
  RT-Thread online packages --->
  AARCH64 qemu virt64 configs --->
```

- ## 2. 使用Smart内核

```

(Top) → RT-Thread Kernel
klibc options --->
(16) The maximal size of kernel object name
[ ] Use the data types defined in ARCH_CPU
[ ] Enable RT-Thread Nano
[*] Enable RT-Thread Smart (microkernel on kernel/userland)
[ ] Enable AMP (Asymmetric Multi-Processing)
[*] Enable SMP (Symmetric multiprocessing)
(4) Number of CPUs
(8) Alignment size for CPU architecture data access
    The maximal level value of priority of thread (32) --->
(100) Tick frequency, Hz
[*] Using stack overflow checking
-* Enable system hook
[*] Using function pointers as system hook
[ ] Enable hook list
-* Enable IDLE Task hook
(4) The max size of idle hook list

```

然后在该目录下执行 `scons` 命令开始编译内核

然后执行当前目录下的 `run.sh` 脚本即可运行rt-smart内核：

```

\ | /
- RT -   Thread Smart Operating System
/ | \   5.2.0 build Jul  6 2025 14:36:07
2006 - 2024 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.
[I/utest] utest is initialize success.
[I/utest] total utest testcase num: (1)
[I/drivers.serial] Using /dev/ttyS0 as default console
Hello RISC-V
msh />

```

编写rust程序

```

→ rustapp git:(main) X tree
.
├── Cargo.lock
├── Cargo.toml
├── build.rs
└── src
    └── lib.rs

```

问题1（已解决）

rust提供平台有限：

```

riscv64-linux-android
riscv64-wrs-vxworks
riscv64gc-unknown-freebsd
riscv64gc-unknown-fuchsia
riscv64gc-unknown-hermit
riscv64gc-unknown-linux-gnu
riscv64gc-unknown-linux-musl
riscv64gc-unknown-netbsd
riscv64gc-unknown-none-elf
riscv64gc-unknown-nutt-elf
riscv64gc-unknown-openbsd
riscv64imac-unknown-none-elf
riscv64imac-unknown-nutt-elf

```

rust不提供rt-smart目标平台，所以尝试了 `riscv64gc-unknown-linux-musl` 和

`riscv64imac-unknown-none-elf` 两个工具链。

先使用 `riscv64gc-unknown-linux-musl`：

1. rt-smart平台不支持硬件浮点，所以会报错：`can't link double-float modules with soft-float modules`。更换 `riscv64imac-unknown-none-elf` 修复。
2. 由于**新版本的GCC编译器**根据SPEC做了更改，将csr相关指令从基本指令集中挪了出去，变成了 `zicsr` 拓展。而RV Linux内核涉及很多csr的指令，如果在编译时没有体现 `zicsr` 拓展，在编译时会报错：

```

march=rv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_zicsr2p0_zifencei2p0_zmmul1p0_zaamo1p0_zalrsc1p0: unsupported ISA subset z`

```

我查看了rust源码中 `riscv64imac-unknown-none-elf` 工具链的支持：

```

options: TargetOptions {
  linker_flavor: LinkerFlavor::Gnu(Cc::No, Lld::Yes),
  linker: Some("rust-lld".into()),
  cpu: "generic-rv64".into(),
  max_atomic_width: Some(64),
  features: "+m,+a,+c".into(),
  llvm_abiname: "lp64".into(),
  panic_strategy: PanicStrategy::Abort,
  relocation_model: RelocModel::Static,
  code_model: Some(CodeModel::Medium),
  emit_debug_gdb_scripts: false,
}

```

配置 `./cargo/config.toml` 文件：

```

[target.riscv64imac-unknown-none-elf]
rustflags = [
  "-C", "target-feature=+i,-zaamo,-zalrsc,-zicsr,-zifencei,-zmmul", # 禁用
zicsr和zifencei扩展
  "-C", "link-arg=-march=rv64imac", # 显式指定工具链支持的架构
  "-C", "link-arg=-mabi=lp64", # 匹配工具链的ABI (lp64)
  "-C", "link-arg=-Wl,--no-relax" # 禁用链接器自动放松优化
]

```

构建 rustapp 后，将生成的.a文件移到 /machines/qemu-virt-riscv64/applications，修改

SConscript.py 添加链接，重新编译即可成功：

```
● → qemu-virt-riscv64 git:(main) X scons -j6
scons: Reading SConscript files ...
Musl version: unknown
scons: done reading SConscript files.
scons: Building targets ...
scons: building associated VariantDir targets: build
scons: `.` is up to date.
scons: done building targets.
```

问题2（未解决）

链接的库会使内核启动卡死：

```
Boot HART ID           : 0
Boot HART Domain       : root
Boot HART Priv Version  : v1.12
Boot HART Base ISA     : rv64imafdc
Boot HART ISA Extensions : time,sstc
Boot HART PMP Count     : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count    : 16
Boot HART MIDELEG       : 0x0000000000001666
Boot HART MEDELEG       : 0x0000000000f0b509
(q)(q)(q)(q)(q)(q)(q)(q)QEMU: Terminated
```

猜测是build.rs写的有问题，所以参考已经实现的build.rs进行构建：

```
// 主要绑定头文件
let bindings = bindgen::Builder::default()
    // 指定主头文件
    .header(format!("{}/include/rththread.h", rththread_path))
    // 使用 core 而非 std
    .use_core()
    // 合并 extern "C" 块
    .merge_extern_blocks(true)
    // 不在 enum 前面添加前缀
    .prepend_enum_name(false)
    // 关闭布局测试
    .layout_tests(false)
    // 添加额外的 clang 参数 (include 路径)
    .clang_arg(format!("-I{}", hw_target_path))
    .clang_arg(format!("-I{/include", rththread_path))
    .clang_arg(format!("-I{/components/legacy", rththread_path))
    .clang_arg(format!("-I{/components/drivers/include", rththread_path))
    .clang_arg(format!("-I{/components/finsh", rththread_path))
    // .clang_arg(format!("-I{/components/libc/compilers/common/extension",
rththread_path))
    // .clang_arg(format!("-I{/components/libc/compilers/common/include/sys/",
rththread_path))
    .clang_arg("-I/home/fox/tools/gcc-arm-none-eabi-5_4-2016q3/arm-none-
eabi/include")
```

```

.clang_arg("-I/home/fox/tools/gcc-arm-none-eabi-5_4-2016q3/arm-none-eabi/include/sys")
.clang_arg(format!("-I{}", cpu_path))
.clang_arg(format!("-I{}", cross_path))
// 生成绑定
.generate()
.expect("Unable to generate bindings");

```

编译 thumbv7em-none-eabihf 平台成功，但是尝试 riscv64imac-unknown-none-elf 平台时失败：

```

error: failed to run custom build command for `rtt_rs2 v0.0.1 (/home/fox/OSPP/rtt_rust/rtt_rs2)`
Caused by:
  process didn't exit successfully: `/home/fox/OSPP/rtt_rust/rtt_rs2/target/debug/build/rtt_rs2-dfcbfab9514d1c7d/build-script-build` (exit status: 101)
  --- stdout
  cargo:rerun-if-changed=wrapper.h
  --- stderr
  /opt/rtt/rt-thread/include/rttypes.h:20:10: fatal error: 'sys/types.h' file not found

thread 'main' panicked at build.rs:35:10:
Unable to generate bindings: ClangDiagnostic("/opt/rtt/rt-thread/include/rttypes.h:20:10: fatal error: 'sys/types.h' file not found\n")
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace

```

我将工具链的库导入后，遇到了非常复杂的问题：

```

--- stderr
/opt/riscv64-linux-musleabi_for_x86_64-pc-linux-gnu/riscv64-unknown-linux-musl/include/sys/errno.h:112: warning: redirecting incorrect #include <sys/errno.h> to <errno.h> [-Wwarnings]
/home/fox/OSPP/riscv_rtt_rust/rt-thread/include/ratomic.h:218:54: warning: incompatible pointer types passing 'rt_atomic_t' (aka '_Atomic(rt_base_t)') to parameter of type 'rt_base_t *' (aka 'long *') [-Wincompatible-pointer-types]
/home/fox/OSPP/riscv_rtt_rust/rt-thread/include/ratomic.h:225:50: error: invalid operands to binary expression ('rt_atomic_t' (aka '_Atomic(rt_base_t)') and 'rt_atomic_t')
clang diag: /opt/riscv64-linux-musleabi_for_x86_64-pc-linux-gnu/riscv64-unknown-linux-musl/include/sys/errno.h:112: warning: redirecting incorrect #include <sys/errno.h> to <errno.h> [-Wwarnings]
clang diag: /home/fox/OSPP/riscv_rtt_rust/rt-thread/include/ratomic.h:218:54: warning: incompatible pointer types passing 'rt_atomic_t *' (aka '_Atomic(rt_base_t)') to parameter of type 'rt_base_t *' (aka 'long *') [-Wincompatible-pointer-types]

thread 'main' panicked at build.rs:54:10:
Unable to generate bindings: ClangDiagnostic("/home/fox/OSPP/riscv_rtt_rust/rt-thread/include/ratomic.h:225:50: error: invalid operands to binary expression ('rt_atomic_t' (aka '_Atomic(rt_base_t)') and 'rt_atomic_t')\n")
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace

```

错误信息 `invalid operands to binary expression ('rt_atomic_t' and 'rt_atomic_t')` 表明编译器无法直接对原子类型 (`rt_atomic_t`) 进行标准运算符操作 (如 `+`、`-` 等)，后续需要查阅更多资料寻找解决方案。（`{RTT_ROOT}/libcpu/risc-v/` 下有很多架构，我都有尝试，但是最终都是以上的问题。）