

2024.03.10-2024.03.16-work-log

工作进展

本阶段主要完成的任务有：选定题目、报名、与项目导师沟通、收集查阅题目的相关资料、搭建项目的开发环境、编译并运行了C语言编写的用户态 `userapps`

经过与项目导师的沟通，我们制定了这样的技术路线来达成题目的目标：

首先，将 `musl libc` 和 `rt-thread` 中所需的C库变量和函数封装到 `libc` 中供用Rust编写的应用程序调用。

其次，制作一个到 `aarch64-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 `aarch64` 平台上的 `rt-smart` 操作系统上运行的指令序列。

最后，我们将选择Rust生态中一些实现与平台有关，需要特定操作系统功能支持的库，为其添加目标为 `rt-smart` 操作系统时的实现代码。

资料收集

`rt-thread` 系统 API： <https://www.rt-thread.org/document/api/index.html>，该 API 为 `rt-smart` 的用户态应用程序可以调用的 API，可以借此实现各种系统功能

Rust FFI 编程： <https://cloud.tencent.com/developer/article/1620862>，通过封装 `libc` 的方式让Rust语言编写的程序能够调用C语言库函数

查看了Rust编译器的源码和其文档，了解了Rust编译器支持的目标平台及其实现： <https://doc.rust-lang.org/nightly/rustc/platform-support.html>

搭建开发环境

安装 `qemu-system-aarch64`

我们选择的目标平台为 `aarch64`，因此我们首先需要安装 `qemu-system-aarch64`，用于支持 `rt-smart` 内核

```
sudo apt install qemu-system-aarch64
```

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：

```
# aarch64 musl gcc
export RTT_CC=gcc
export RTT_EXEC_PATH=/yourpath/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/bin
export RTT_CC_PREFIX=aarch64-linux-musleabi-
export PATH=$PATH:$RTT_EXEC_PATH
```

使用命令 `source ~/.bashrc` 刷新环境变量配置文件

之后可使用命令 `aarch64-linux-musleabi-gcc -v` 检查 `musl gcc` 工具环境变量是否正确设置

```
diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021:~$ aarch64-linux-musleabi-gcc -v
使用内建 specs。
COLLECT_GCC=aarch64-linux-musleabi-gcc
COLLECT_LTO_WRAPPER=/opt/aarch64-smart-musleabi/bin/./libexec/gcc/aarch64-linux-musleabi/12.2.0/lto-wrapper
目标: aarch64-linux-musleabi
配置为: ../src/gcc/configure --disable-werror --prefix= --target=aarch64-linux-musleabi --with-sysroot=/aarch64-linux-musleabi --with-build-sysroot=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/sysroot/ --enable-languages=c,c++ --disable-multilib --enable-tls --disable-libmudflap --disable-libsanitizer --disable-gnu-indirect-function --disable-libmpx --enable-libstdcxx-time --host=x86_64-pc-linux-gnu AR_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/ar AS_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/gas/as-new LD_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/ld/ld-new NM_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/nm-new OBJCOPY_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/objcopy OBJDUMP_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/objdump RANLIB_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/ranlib READELF_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/readelf STRIP_FOR_TARGET=/builds/alliance/rt-smart/musl-toolchain/build/aarch64-linux-musleabi_for_x86_64-pc-linux-gnu/obj_binutils/binutils/strip-new
线程模型: posix
Supported LTO compression algorithms: zlib
gcc 版本 12.2.0 (GCC)
build date: Oct 20 2023 16:25:01
musl sha: 15706d647d27218f1735edce5eec28f50fedee22
build sha: 15706d647d27218f1735edce5eec28f50fedee22
build job: 547559
build pipeline: 203958
```

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

首先将rt-smart的源码下载到本地: <https://github.com/RT-Thread/rt-thread.git>

进入到 `qemu-virt64-aarch64` 目录下

```
cd ./rt-thread/bsp/qemu-virt64-aarch64/ #打开 rt-thread 项目目录中的 bsp/qemu-virt64-aarch64 目录
scons --menuconfig
```

1. 选择RT-Thread Kernel选项

RT-Thread Project Configuration

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in []

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 --->

<Select>

< Exit >

< Help >

< Save >

< Load >

2. 使用Smart内核

> RT-Thread Kernel

RT-Thread Kernel

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in []

(16) The maximal size of kernel object name

[] Use the data types defined in ARCH_CPU

[*] Enable RT-Thread Smart (microkernel on kernel/userland)

[] Enable RT-Thread Nano

[] 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

v(+)

<Select>

< Exit >

< Help >

< Save >

< Load >

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 `qemu.sh` 脚本即可运行rt-smart内核

```
+ diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021: ...
diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021:~/os-educg/rt-thread/bsp/qemu-virt64-aarch64$ ./qemu.sh
[I/rtdm.ofw] Booting RT-Thread on physical CPU 0x0
[I/rtdm.ofw] Machine model: linux,dummy-virt
[I/rtdm.ofw] Memory node(0) ranges: 0x0000000040000000 - 0x0000000048000000
[I/cpu.aa64] Reserved memory:
[I/cpu.aa64]   kernel      [0x0000000040080000, 0x00000000401befd0]
[I/cpu.aa64]   fdt        [0x00000000441befd0, 0x00000000442befd0]
[I/cpu.aa64]   memheap    [0x00000000401befd0, 0x00000000441befd0]
[I/cpu.aa64] Usable memory:
[I/cpu.aa64]   memory@40000000 [0x0000000040000000, 0x0000000040080000]
[I/cpu.aa64]   memory@40000000 [0x00000000442befd0, 0x0000000048000000]
[I/osi.psci] Using PSCI v0.2 Function IDs
[I/rtdm.ofw] Console: uart0 (<no-node>)

\ | /
- RT -   Thread Smart Operating System
/ | \    5.1.0 build Mar  7 2024 10:31:29
2006 - 2024 Copyright by RT-Thread team
file system initialization done!
hello rt-thread
msh />
```

编译并运行用户态userapps

克隆仓库

将Smart的userapps仓库克隆下来:

```
git clone https://github.com/RT-Thread/userapps.git
```

编译

首先运行env.sh 添加一下环境变量

```
source env.sh
```

进入 apps 目录进行编译

```
cd apps
xmake f -a aarch64 # 配置为 aarch64平台
xmake -j8
```

镜像制作

运行 `xmake smart-rootfs` 制作 rootfs，运行 `xmake smart-image` 制作镜像

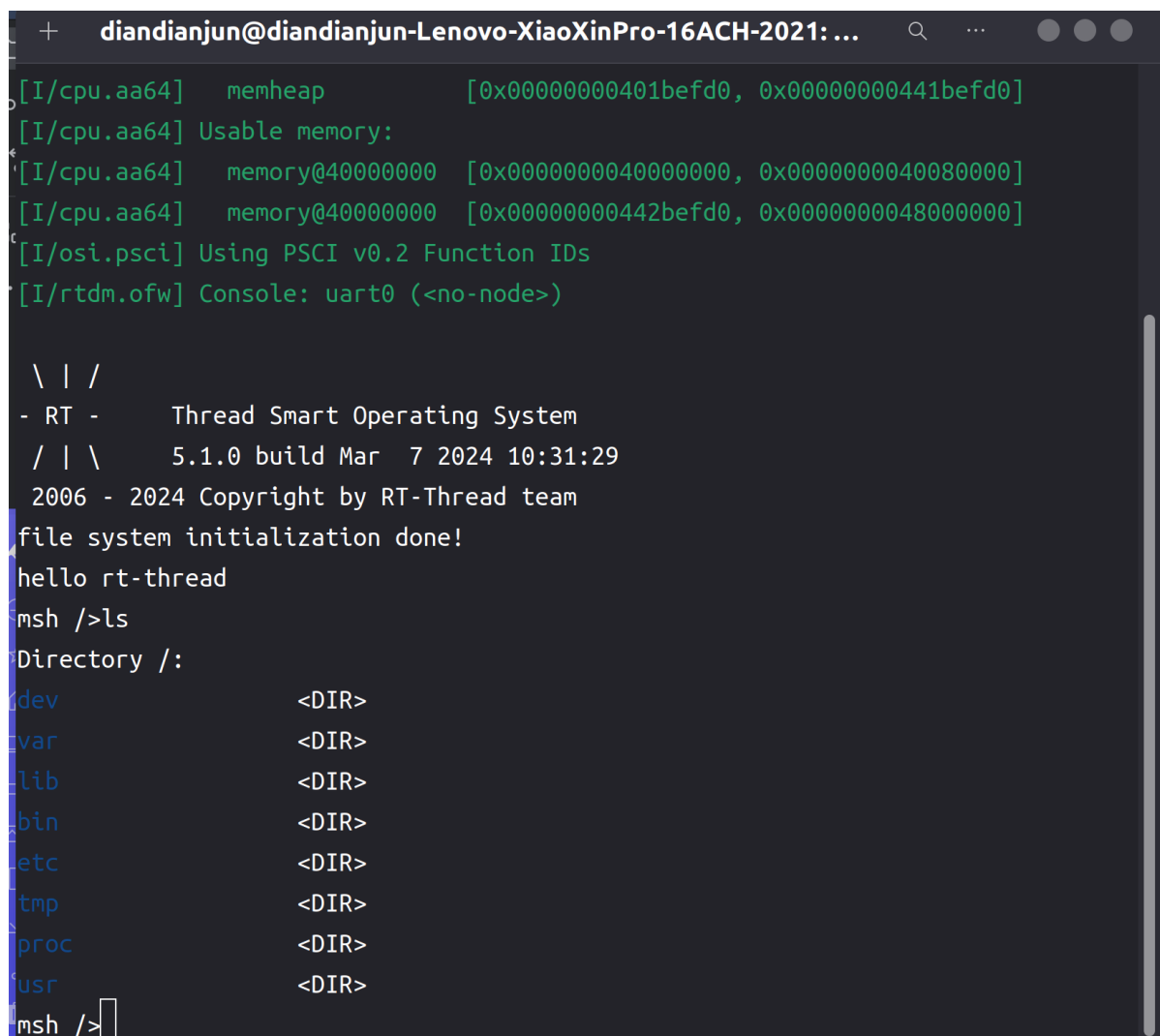
```
xmake smart-rootfs
xmake smart-image -f fat -o ../prebuilt/qemu-virt64-aarch64-fat/fat.img #制作 fat
镜像
```

之后将 `fat.img`，`rtthread.bin`，`qemu.sh` 放在一个目录下，将 `qemu.sh` 中的 `file=sd.bin` 修改为 `file=fat.img`，然后运行 `qemu.sh`

使用命令

```
ls
```

可以看到文件系统内的一些文件



```
+ diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021: ...
[I/cpu.aa64] memheap [0x00000000401befd0, 0x00000000441befd0]
[I/cpu.aa64] Usable memory:
[I/cpu.aa64] memory@40000000 [0x0000000040000000, 0x0000000040080000]
[I/cpu.aa64] memory@40000000 [0x00000000442befd0, 0x0000000048000000]
[I/osi.psci] Using PSCI v0.2 Function IDs
[I/rtdm.ofw] Console: uart0 (<no-node>)

\ | /
- RT - Thread Smart Operating System
/ | \ 5.1.0 build Mar 7 2024 10:31:29
2006 - 2024 Copyright by RT-Thread team
file system initialization done!
hello rt-thread
msh />ls
Directory /:
dev <DIR>
var <DIR>
lib <DIR>
bin <DIR>
etc <DIR>
tmp <DIR>
proc <DIR>
usr <DIR>
msh />
```

运行命令：

```
cd bin
ls
```

可以看到用户态编写的应用程序和系统里的一些命令程序混在一起

```
diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021: ...
msh />cd bin
msh /bin>ls
Directory /bin:
kill 1116488
date 1116488
libpng16-config 2392
mkdir 1116488
stty 1116488
cp 1116488
setserial 1116488
uname 1116488
lsattr 1116488
chgrp 1116488
usleep 1116488
png-fix-ixt 179136
ed 1116488
login 1116488
mknod 1116488
su 1116488
pwd 1116488
chown 1116488
link 1116488
vi 1116488
getopt 1116488
```

可以找到之前apps里的hello应用程序在其中

```
diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021: ...  
ffprobe      17315400  
ash          1116488  
ipcalc       1116488  
pidof        1116488  
resume       1116488  
gunzip       1116488  
rmdir        1116488  
dnsdomainname 1116488  
nuke         1116488  
minips       1116488  
netstat      1116488  
sh           1116488  
hello        66816  
bash         1116488  
sleep        1116488  
jpegtran     485344  
lzop         1116488  
mv           1116488  
fgrep        1116488  
mktemp       1116488  
egrep        1116488  
fatattr      1116488  
djpeg        487096  
uncompress   1116488
```

运行命令：

```
./hello
```

可以观察到输出hello world

```
diandianjun@diandianjun-Lenovo-XiaoXinPro-16ACH-2021: ...  
shm_ping          132616  
chattr            1116488  
df                1116488  
arch              1116488  
setarch           1116488  
watch             1116488  
shm_pong          132616  
linux32           1116488  
fdflush           1116488  
more              1116488  
bbconfig           1116488  
fsync             1116488  
zcat              1116488  
tar               1116488  
true              1116488  
libpng-config     2392  
thread_example    132616  
gzip              1116488  
player            15309704  
pipe_progress     1116488  
ps                1116488  
msh /bin>./hello  
msh /bin>hello world!  
[
```