DragonOS

发布 *dev*

longjin

2024 年 04 月 27 日

入门

CHAPTER 1

DragonOS 简介

DragonOS 龙操作系统是一个面向云计算轻量化场景的，完全自主内核的，提供 Linux 二进制兼容性的 64 位操作系统。它使用 Rust 语言进行开发，以提供更好的可靠性。目前在 Rust 操作系统领域，DragonOS 在 Github 排行全国稳居前三位。

DragonOS 开源社区成立于 2022 年 7 月，它完全商业中立。我们的目标是，构建一个完全独立自主的、开源的、高性能及高可靠性的服务器操作系统，**打造完全自主可控的数字化未来！**

DragonOS 具有优秀的、完善的架构设计。相比于同体量的其他系统，DragonOS 支持虚拟化，并在设备模型、调度子系统等方面具有一定优势。当前正在大力推进云平台支持、riscv 支持等工作，以及编译器、应用软件的移植。力求在 5 年内实现生产环境大规模应用。

DragonOS 目前在社区驱动下正在快速发展中，目前 DragonOS 已经实现了约 1/4 的 Linux 接口，在未来我们将提供对 Linux 的 100% 兼容性，并且提供新特性。

DragonOS 的目标是，构建一个完全独立自主的、开源的、高性能及高可靠性的服务器操作系统，为国家数字基础设施建设提供完全独立自主的底层核心动力。

作为一个社区驱动的开源操作系统，为了促进开源社区建设，并避免让其遭受一些不遵守开源协议的商业公司的侵权，我们决定使用 GPLv2 协议开放源代码，以严格的开源协议来保护 DragonOS。

你可能对 DragonOS 中已经实现了哪些功能感兴趣，您可以转到这里：功能特性

1.1 DragonOS 的功能

1.1.1 规范

- 启动引导：Multiboot2
- 接口：posix 2008

1.1.2 内核层

内存管理

- 页帧分配器
- 小对象分配器
- VMA
- MMIO 地址空间自动分配
- 页面映射器
- 硬件抽象层
- 独立的用户地址空间管理机制
- C 接口兼容层

多核

- 多核引导
- ipi 框架

进程管理

- 进程创建
- 进程回收
- 内核线程
- fork
- exec
- 进程睡眠（支持高精度睡眠）
- kthread 机制

- 可扩展二进制加载器

同步原语

- mutex 互斥量
- semaphore 信号量
- atomic 原子变量
- spinlock 自旋锁
- wait_queue 等待队列

调度

- CFS 调度器
- 实时调度器（FIFO、RR）
- 单核调度
- 多核调度
- 负载均衡

IPC

- 匿名 pipe 管道
- signal 信号

文件系统

- VFS
- fat12/16/32
- Devfs
- RamFS
- Procfs
- Sysfs

异常及中断处理

- APIC
- softirq 软中断
- 内核栈 traceback

内核实用库

- 字符串操作库
- ELF 可执行文件支持
- printk
- 基础数学库
- 屏幕管理器
- textui 框架
- CRC 函数库
- 通知链

系统调用

请见系统调用文档

测试框架

- ktest

驱动程序

- ACPI 高级电源配置模块
- IDE 硬盘
- AHCI 硬盘
- PCI、PCIe 总线
- XHCI（usb3.0）
- ps/2 键盘
- ps/2 鼠标
- HPET 高精度定时器

- RTC 时钟
- local apic 定时器
- UART 串口
- VBE 显示
- VirtIO 网卡
- x87FPU
- TTY 终端
- 浮点处理器

1.1.3 用户层

LibC

- 基础系统调用
- 基础标准库函数
- 部分数学函数

shell 命令行程序

- 基于字符串匹配的解析
- 基本的几个命令

Http Server

- 使用 C 编写的简单的 Http Server，能够运行静态网站。

1.1.4 软件移植

- GCC 11.3.0 （暂时只支持了 x86_64 的 Cross Compiler）https://github.com/DragonOS-Community/gcc
- binutils 2.38（暂时只支持了 x86_64 的 Cross Compiler）https://github.com/DragonOS-Community/binutils
- gmp 6.2.1 https://github.com/DragonOS-Community/gmp-6.2.1
- mpfr 4.1.1 https://github.com/DragonOS-Community/mpfr
- mpc 1.2.1 https://github.com/DragonOS-Community/mpc
- relibc https://github.com/DragonOS-Community/relibc
- sqlite3

CHAPTER 2

构建 DragonOS

2.1 1. 写在前面

无论您采用后文中的何种方式来编译 DragonOS，您必须先按照本小节中的步骤，初始化您的开发环境。

开始之前，您需要一台运行 Linux 或 MacOS 的计算机，并且处理器架构为 X86-64.

对于 Linux 发行版，建议使用 Ubuntu22、Debian、Arch Linux 这样的，仓库软件版本较新的发行版，这能为您减少很多麻烦。

2.1.1 1.1 下载 DragonOS 的源代码

使用 https 克隆：

```
git clone https://github.com/DragonOS-Community/DragonOS.git
cd DragonOS
# 使用镜像源更新子模块
make update-submodules-by-mirror
```

为了方便后续的开发，我们建议您使用 ssh 克隆（请先配置好 github 的 SSH Key），以避免由于网络问题导致的克隆失败：

使用 ssh 克隆（请先配置好 github 的 SSH Key）：

```
# 使用ssh克隆
git clone git@github.com:DragonOS-Community/DragonOS.git
cd DragonOS
# 使用镜像源更新子模块
make update-submodules-by-mirror
```

2.2 2. 使用一键初始化脚本进行安装（推荐）

我们提供了一键初始化脚本，可以一键安装，只需要在控制台运行以下命令：

```
cd DragonOS
cd tools
bash bootstrap.sh  # 这里请不要加上sudo,␣
↪因为需要安装的开发依赖包是安装在用户环境而非全局环境
```

备注： 一键配置脚本目前只支持以下系统：

- Ubuntu/Debian/Deepin/UOS 等基于 Debian 的衍生版本

欢迎您为其他的系统完善构建脚本！

如果一键初始化脚本能够正常运行，并输出最终的“祝贺”界面 (如下所示)，请关闭当前终端，然后重新打开。

```
|-----------Congratulations!---------------|
|                                          |
|   你成功安装了DragonOS所需的依赖项！          |
|                                          |
|   请关闭当前终端，并重新打开一个终端           |
|   然后通过以下命令运行：                     |
|                                          |
|                make run                  |
|                                          |
|------------------------------------------|
```

接着，请直接跳到编译命令讲解进行阅读！

2.3 3. 手动安装

2.3.1 3.1 依赖清单

如果自动安装脚本不能支持您的操作系统，那么您需要手动安装依赖程序。以下是依赖项的清单：

在以下依赖项中，除了 `docker-ce` 和 `Rust` 及其工具链以外，其他的都能通过系统自带的包管理器进行安装。关于 docker 以及 rust 的安装，请看后文。

- docker-ce
- llvm-dev
- libclang-dev
- clang
- gcc-multilib
- qemu qemu-system qemu-kvm
- build-essential
- fdisk
- lsb-release
- git
- dosfstools
- unzip
- Rust 以及其工具链

请留意，若您的 Linux 系统是在虚拟机中运行的，还请您在您的 VMware/Virtual Box 虚拟机的处理器设置选项卡中，开启 Intel VT-x 或 AMD-V 选项，否则，DragonOS 将无法运行。

备注： *在某些 Linux 发行版的软件仓库中构建的 Qemu 可能由于版本过低而不兼容 DragonOS，如果遇到这种问题，请卸载 Qemu，并采用编译安装的方式重新安装 Qemu*

在该地址下载 Qemu 源代码：https://download.qemu.org/

解压后进入源代码目录，然后执行下列命令：

```
# 安装编译依赖项
sudo apt install -y autoconf automake autotools-dev curl libmpc-dev libmpfr-dev␣
↪libgmp-dev \
              gawk build-essential bison flex texinfo gperf libtool patchutils bc \
              zlib1g-dev libexpat-dev pkg-config  libglib2.0-dev libpixman-1-dev␣
↪libsdl2-dev \
```

(下页继续)

(续上页)

```
            git tmux python3 python3-pip ninja-build

./configure --enable-kvm
make -j 8
sudo make install
# 编译安装完成
```

请注意，编译安装的 QEMU，将通过 VNC 模式进行链接，因此，您还需要在您的计算机上安装 VNC viewer 以连接至 QEMU 虚拟机。

2.3.2 3.2 安装 Docker

您可以在 docker 官网下载安装 docker-ce.

详细信息请转到：https://docs.docker.com/engine/install/

2.3.3 3.3 安装 Rust

警告： **【常见误区】**：如果您打算采用 docker 进行编译，尽管 docker 镜像中已经安装了 Rust 编译环境，但是，为了能够在 VSCode 中使用 Rust-Analyzer 进行代码提示，以及 `make clean` 命令能正常运行，您的客户机上仍然需要安装 rust 环境。

您可以在控制台输入以下命令，安装 rust。

```
# 这两行用于换源，加速Rust的安装过程
export RUSTUP_DIST_SERVER=https://mirrors.ustc.edu.cn/rust-static
export RUSTUP_UPDATE_ROOT=https://mirrors.ustc.edu.cn/rust-static/rustup
# 安装Rust
curl https://sh.rustup.rs -sSf | sh -s -- --default-toolchain nightly
# 把Rustup加到环境变量
echo "export PATH=\"\$HOME/.cargo/bin:\$PATH\"" >> ~/.bashrc
source ~/.cargo/env
source "$HOME/.cargo/env"

# 更换cargo的索引源
touch ~/.cargo/config
echo -e "[source.crates-io]   \n \
registry = \"https://github.com/rust-lang/crates.io-index\"  \n \
\n \
```

(下页继续)

(续上页)

```
replace-with = 'dragonos-gitee' \n \
[source.dragonos-gitee] \n \
registry = \"https://gitee.com/DragonOS/crates.io-index.git\"         \n \
" > ~/.cargo/config

# 安装DragonOS所需的工具链
cargo install cargo-binutils
rustup toolchain install nightly
rustup default nightly
rustup component add rust-src
rustup component add llvm-tools-preview
rustup target add x86_64-unknown-none
# Rust安装完成
```

至此，公共依赖项已经安装完成，您可以根据自己的需要，阅读后续章节

关于编译命令的用法，请见：编译命令讲解

2.4 4. 从 Docker 构建（不推荐）

DragonOS 发布了一个 Docker 编译环境，便于开发者运行 DragonOS。但是，由于编码过程仍需要在客户机上进行，因此，您需要在客户机上安装 Rust 编译环境。

本节假设以下操作均在 Linux 下进行。

2.4.1 4.1 安装 qemu 虚拟机

在本节中，我们建议您采用命令行安装 qemu：

```
sudo apt install -y qemu qemu-system qemu-kvm
```

2.4.2 4.2 创建磁盘镜像

首先，您需要使用 tools 文件夹下的 create_hdd_image.sh，创建一块虚拟磁盘镜像。您需要在 tools 文件夹下运行此命令。

```
bash create_hdd_image.sh
```

2.4.3 4.3 运行 DragonOS

如果不出意外的话，这将是运行 DragonOS 的最后一步。您只需要在 DragonOS 的根目录下方，执行以下命令，即可运行 DragonOS。

```
make run-docker
```

稍等片刻，DragonOS 将会被运行。

在 qemu 虚拟机被启动后，我们需要在控制台输入字母 `c`，然后回车。这样，虚拟机就会开始执行。

备注:

1. 首次编译时，由于需要下载 Rust 相关的索引（几百 MB 大小），因此需要一定的时间，请耐心等候!
2. 输入命令可能需要加上 sudo

关于编译命令的用法，请见：编译命令讲解

2.5 5. 其他注意事项

2.5.1 5.1 创建磁盘镜像

首先，您需要使用**普通用户**权限运行 `tools/create_hdd_image.sh`，为 DragonOS 创建一块磁盘镜像文件。该脚本会自动完成创建磁盘镜像的工作，并将其移动到 `bin/`目录下。 请注意，由于权限问题，请务必使用**普通用户**权限运行此脚本。(运行后，需要提升权限时，系统可能会要求您输入密码)

2.5.2 5.2 编译、运行 DragonOS

1. 安装编译及运行环境
2. 进入 DragonOS 文件夹
3. 输入 `make run` 即可编译并写入磁盘镜像，并运行

在 qemu 虚拟机被启动后，我们需要在控制台输入字母 `c`，然后回车。这样，虚拟机就会开始执行。

备注: 首次编译时，由于需要下载 Rust 相关的索引（几百 MB 大小），因此需要一定的时间，请耐心等候!

关于编译命令的用法，请见：编译命令讲解

2.6 6. 编译命令讲解

- 本地编译，不运行: `make all -j 您的 CPU 核心数`
- 本地编译，并写入磁盘镜像，不运行: `make build`
- 本地编译，写入磁盘镜像，并在 QEMU 中运行: `make run`
- Docker 编译，并写入磁盘镜像,: `make docker`
- Docker 编译，写入磁盘镜像，并在 QEMU 中运行: `make run-docker`
- 不编译，直接从已有的磁盘镜像启动: `make qemu`
- 清理编译产生的文件: `make clean`
- 编译文档: `make docs`（需要手动安装 sphinx 以及 docs 下的 `requirements.txt` 中的依赖）
- 清理文档: `make clean-docs`
- 格式化代码: `make fmt`

备注： 如果您需要在 vnc 中运行 DragonOS，请在上述命令后加上`-vnc`后缀。如：`make run-vnc`

qemu 虚拟机将在 5900 端口监听 vnc 连接。您可以使用 vnc viewer 或者 Remmina 连接至 qemu 虚拟机。

2.7 7. 为 riscv64 编译

由于目前 DragonOS 尚未完全移植到 riscv64，因此编译需要这样做：

1. 修改 `env.mk` 和`.vscode/settings.json`

把 `env.mk` 里面的 `ARCH` 的值改为 `riscv64`，并且在 `setting.json` 里面注释`"rust-analyzer.cargo.target": "x86_64-unknown-none",`，改为启用 riscv64 的那行。

2. 重启 rust-analyzer
3. 清理编译缓存

由于 x86_64 和 riscv64 架构差异，可能存在缓存导致的编译问题，确保运行前先清理缓存。

```
make clean
```

4. 为 riscv64 编译并运行

```
# 下载DragonStub
git submodule update --init --recursive --force

make kernel -j $(nproc) && make write_diskimage && make qemu
```

请注意，由于是在控制台运行 qemu，当你想要退出的时候，输入 `Ctrl+A` 然后按 `X` 即可。

CHAPTER 3

DragonOS 镜像站

您可以从以下镜像站下载到 DragonOS 的源代码和其他文件：

- DragonOS 镜像站 https://mirrors.dragonos.org/
- DragonOS 国内镜像站 (RinGoTek)
- git 镜像站

CHAPTER 4

内核编译配置

4.1 内核编译配置说明

4.1.1 原理

在内核目录下，用 kernel.config 来设置内核编译配置信息，以类似解析 toml 文件的方式去解析该文件，然后接着去解析各模块下的 d.config 以获取 feature 的启用情况

4.1.2 示例

kernel.config

```
[[module.include]]
name = "init"
path = "src/init/"
enable = "y"
description = ""

[[module.include]]
name = "mm"
path = "src/mm/"
enable = "y"
description = ""
```

- **[[module.include]]:** 将模块加入到 include 列表中
- **name:** 模块名
- **path:** 模块路径，存放着 d.config
- **enable:**
  - **y:** 启用，解析模块下的 d.config
  - **n:** 不启用，不解析
- **description:** 模块的描述信息

src/mm/d.config

```
[module]
name = "mm"
description = ""

[[module.include]]
name = "allocator"
path = "src/mm/allocator/"
enable = "y"
description = ""

[[module.features]]
name = "mm_debug"
enable = "y"
description = ""
```

- **[module]:** 当前模块
  - **name:** 当前模块名称
  - **description:** 模块的描述信息
- **[[module.include]]:** 当前模块下所包含的模块，与 kernel.config 下的相同
- **[[module.features]]:** 当前模块下的 feature
  - **name:** feature 名
  - **enable:** 是否开启
    - **y:** 开启
    - **n:** 不开启
  - **description:** feature 的描述信息

以下是其它模块下的 d.config:

src/mm/allocator/d.config

```
[module]
name = "allocator"
description = ""

[[module.features]]
name = "allocator_debug"
enable = "y"
description = ""
```

src/init/d.config

```
[module]
name = "init"
description = ""

[[module.features]]
name = "init_debug"
enable = "y"
description = ""
```

上面所有已开启模块的 d.config 中的 feature，会最终生成到内核目录下的 D.config 文件，即 D.config 是最终内核编译的配置，如下：

D.config

```
init_debug = y
allocator_debug = y
mm_debug = y
```

4.2 目标架构配置

4.2.1 支持的架构

- x86_64
- riscv64

4.2.2 架构相关配置

为了能支持 vscode 的调试功能，我们需要修改.vscode/settings.json 文件的以下行：

```
    "rust-analyzer.cargo.target": "riscv64gc-unknown-none-elf",
    // "rust-analyzer.cargo.target": "x86_64-unknown-none",
```

如果想要为 x86_64 架构编译，请启用 x86_64 那一行，注释掉其它的。如果想要为 riscv64 架构编译，请启用 riscv64 那一行，注释掉其它的。

同时，我们还需要修改 makefile 的环境变量配置：

请修改 env.mk 文件的以下行：

```
ifeq ($(ARCH), )
# ！！！！！ 在这里设置ARCH，可选x86_64和riscv64
# !!!!!!!如果不同时调整这里以及vscode的settings.json，那么自动补全和检查将会失效
export ARCH=riscv64
endif
```

请注意，更换架构需要重新编译，因此请运行 make clean 清理编译结果。然后再运行 make run 即可。

CHAPTER 5

引导加载

DragonOS 采用 GRUB2 作为其引导加载程序，支持 Multiboot2 协议引导。目前仅支持 GRUB2.06 版本。

5.1 引导加载程序

5.1.1 原理

目前，DragonOS 支持 Legacy BIOS 以及 UEFI 两种方式，进行启动引导。

在 `head.S` 的头部包含了 Multiboot2 引导头，里面标志了一些 Multiboot2 相关的特定信息，以及一些配置命令。

在 DragonOS 的启动初期，会存储由 GRUB2 传来的 magic number 以及 multiboot2_boot_info_addr。当系统进入 `Start_Kernel` 函数之后，将会把这两个信息保存到 multiboot2 驱动程序之中。信息的具体含义请参照 Multiboot2 Specification 进行理解，该部分难度不大，相信读者经过思考能理解其中的原理。

5.1.2 参考资料

- Multiboot2 Specification
- GNU GRUB Manual 2.06
- UEFI/Legacy 启动 - yujianwu - DragonOS 社区

5.2 Multiboot2 支持模块

Multiboot2 支持模块提供对 Multiboot2 协议的支持。位于 `kernel/driver/multiboot2` 文件夹中。

根据 Multiboot2 协议，操作系统能够从 BootLoader 处获得一些信息，比如基本的硬件信息以及 ACPI 表的起始地址等。

5.2.1 数据结构

`kernel/driver/multiboot2/multiboot2.h` 中按照 Multiboot2 协议的规定，定义了大部分的数据结构，具体细节可查看该文件: DragonOS/multiboot2.h at master · fslongjin/DragonOS · GitHub

---

5.2.2 迭代器

由于 Multiboot2 的信息存储在自 `multiboot2_boot_info_addr` 开始的一段连续的内存空间之中，且不同类型的 header 的长度不同，因此设计了一迭代器 `multiboot2_iter`。

函数原型

```
void multiboot2_iter(bool (*_fun)(const struct iter_data_t *, void *, unsigned int *),
                     void *data, unsigned int *count)
```

_fun

指定的 handler。当某个 header 的 tag 与该 handler 所处理的 tag 相同时，handler 将处理该 header，并返回 true。

其第一个参数为 tag 类型，第二个参数为返回的数据的指针，第三个值为计数（某些没有用到该值的地方，该值可以为空）

data

传递给 `_fun` 的第二个参数，`_fun` 函数将填充该指针所指向的内存区域，从而返回数据。

count

当返回的 **data** 为一个列表时，通过该值来指示列表中有多少项。

5.2.3 迭代工作函数

在模块中，按照我们需要获取不同类型的 tag 的需要，定义了一些迭代器工作函数。

CHAPTER 6

核心 API 文档

这里是 DragonOS 的核心 api 文档。

6.1 DragonOS 内核核心 API

6.1.1 循环链表管理函数

循环链表是内核的重要的数据结构之一。包含在 `kernel/common/list.h` 中。

`void list_init(struct List *list)`

描述

初始化一个 List 结构体，使其 prev 和 next 指针指向自身

参数

list

要被初始化的 List 结构体

```
void list_add(struct List *entry, struct List *node)
```

描述

将 node 插入到 entry 的后方

参数

entry

已存在于循环链表中的一个结点

node

待插入的结点

```
void list_append(struct List *entry, struct List *node)
```

描述

将 node 插入到 entry 的前方

参数

entry

已存在于循环链表中的一个结点

node

待插入的结点

```
void list_del(struct List *entry)
```

描述

从链表中删除结点 entry

参数

entry

待删除的结点

```
list_del_init(struct List *entry)
```

描述

从链表中删除结点 entry，并将这个 entry 使用 list_init() 进行重新初始化。

参数

entry

待删除的结点

```
bool list_empty(struct List *entry)
```

描述

判断链表是否为空

参数

entry

链表中的一个结点

```
struct List *list_prev(struct List *entry)
```

描述

获取 entry 的前一个结点

参数

entry

链表中的一个结点

```
struct List *list_next(struct List *entry)
```

描述

获取 entry 的后一个结点

参数

entry

链表中的一个结点

```
void list_replace(struct List *old, struct List *new)
```

描述

将链表中的 old 结点替换成 new 结点

参数

old

要被换下来的结点

new

要被换入链表的新的结点

`list_entry(ptr, type, member)`

描述

该宏能通过 ptr 指向的 List 获取到 List 所处的结构体的地址

参数

ptr

指向 List 结构体的指针

type

要被换入链表的新的结点

member

List 结构体在上述的“包裹 list 结构体的结构体”中的变量名

`list_first_entry(ptr, type, member)`

描述

获取链表中的第一个元素。请注意，该宏要求链表非空，否则会出错。

参数

与*list_entry()*相同

`list_first_entry_or_null(ptr, type, member)`

描述

获取链表中的第一个元素。若链表为空，则返回 NULL。

参数

与*list_entry()*相同

list_last_entry(ptr, type, member)

描述

获取链表中的最后一个元素。请注意，该宏要求链表非空，否则会出错。

参数

与*list_entry()*相同

list_last_entry_or_full(ptr, type, member)

描述

获取链表中的最后一个元素。若链表为空，则返回 NULL。

参数

与*list_entry()*相同

list_next_entry(pos, member)

描述

获取链表中的下一个元素

参数

pos

指向当前的外层结构体的指针

member

链表结构体在外层结构体内的变量名

list_prev_entry(pos, member)

描述

获取链表中的上一个元素

参数

与*list_next_entry()*相同

list_for_each(ptr, head)

描述

遍历整个链表（从前往后）

参数

ptr

指向 List 结构体的指针

head

指向链表头结点的指针 (struct List*)

list_for_each_prev(ptr, head)

描述

遍历整个链表（从后往前）

参数

与*list_for_each()*相同

list_for_each_safe(ptr, n, head)

描述

从前往后遍历整个链表（支持删除当前链表结点）

该宏通过暂存中间变量，防止在迭代链表的过程中，由于删除了当前 ptr 所指向的链表结点从而造成错误.

参数

ptr

指向 List 结构体的指针

n

用于存储临时值的 List 类型的指针

head

指向链表头结点的指针 (struct List*)

list_for_each_prev_safe(ptr, n, head)

描述

从后往前遍历整个链表.（支持删除当前链表结点）

该宏通过暂存中间变量，防止在迭代链表的过程中，由于删除了当前 ptr 所指向的链表结点从而造成错误.

参数

与*list_for_each_safe()*相同

list_for_each_entry(pos, head, member)

描述

从头开始迭代给定类型的链表

参数

pos

指向特定类型的结构体的指针

head

指向链表头结点的指针 (struct List*)

member

struct List 在 pos 的结构体中的成员变量名

`list_for_each_entry_reverse(pos, head, member)`

描述

逆序迭代给定类型的链表

参数

与*list_for_each_entry()*相同

`list_for_each_entry_safe(pos, n, head, member)`

描述

从头开始迭代给定类型的链表（支持删除当前链表结点）

参数

pos

指向特定类型的结构体的指针

n

用于存储临时值的，和 pos 相同类型的指针

head

指向链表头结点的指针 (struct List*)

member

struct List 在 pos 的结构体中的成员变量名

list_prepare_entry(pos, head, member)

描述

为*list_for_each_entry_continue()*准备一个’pos’结构体

参数

pos

指向特定类型的结构体的，用作迭代起点的指针

head

指向要开始迭代的 struct List 结构体的指针

member

struct List 在 pos 的结构体中的成员变量名

list_for_each_entry_continue(pos, head, member)

描述

从指定的位置的【下一个元素开始】，继续迭代给定的链表

参数

pos

指向特定类型的结构体的指针。该指针用作迭代的指针。

head

指向要开始迭代的 struct List 结构体的指针

member

struct List 在 pos 的结构体中的成员变量名

list_for_each_entry_continue_reverse(pos, head, member)

描述

从指定的位置的【上一个元素开始】,【逆序】迭代给定的链表

参数

与*list_for_each_entry_continue()*的相同

list_for_each_entry_from(pos, head, member)

描述

从指定的位置开始, 继续迭代给定的链表

参数

与*list_for_each_entry_continue()*的相同

list_for_each_entry_safe_continue(pos, n, head, member)

描述

从指定的位置的【下一个元素开始】, 继续迭代给定的链表.（支持删除当前链表结点）

参数

pos

指向特定类型的结构体的指针。该指针用作迭代的指针。

n

用于存储临时值的，和 pos 相同类型的指针

head

指向要开始迭代的 struct List 结构体的指针

member

struct List 在 pos 的结构体中的成员变量名

`list_for_each_entry_safe_continue_reverse(pos, n, head, member)`

描述

从指定的位置的【上一个元素开始】,【逆序】迭代给定的链表。(支持删除当前链表结点)

参数

与*list_for_each_entry_safe_continue()*的相同

`list_for_each_entry_safe_from(pos, n, head, member)`

描述

从指定的位置开始, 继续迭代给定的链表. (支持删除当前链表结点)

参数

与*list_for_each_entry_safe_continue()*的相同

6.1.2 基础 C 函数库

内核编程与应用层编程不同，你将无法使用 LibC 中的函数来进行编程。为此，内核实现了一些常用的 C 语言函数，并尽量使其与标准 C 库中的函数行为相近。值得注意的是，这些函数的行为可能与标准 C 库函数不同，请在使用时仔细阅读以下文档，这将会为你带来帮助。

字符串操作

`int strlen(const char *s)`

描述

测量并返回字符串长度。

参数

src

源字符串

```
long strnlen(const char *src, unsigned long maxlen)
```

描述

测量并返回字符串长度。当字符串长度大于 maxlen 时，返回 maxlen

参数

src

源字符串

maxlen

最大长度

```
long strnlen_user(const char *src, unsigned long maxlen)
```

描述

测量并返回字符串长度。当字符串长度大于 maxlen 时，返回 maxlen。

该函数会进行地址空间校验，要求 src 字符串必须来自用户空间。当源字符串来自内核空间时，将返回 0.

参数

src

源字符串，地址位于用户空间

maxlen

最大长度

```
char *strncpy(char *dst, const char *src, long count)
```

描述

拷贝长度为 count 个字节的字符串，返回 dst 字符串

参数

src

源字符串

dst

目标字符串

count

要拷贝的源字符串的长度

```
char *strcpy(char *dst, const char *src)
```

描述

拷贝源字符串，返回 dst 字符串

参数

src

源字符串

dst

目标字符串

```
long strncpy_from_user(char *dst, const char *src, unsigned long size)
```

描述

从用户空间拷贝长度为 count 个字节的字符串到内核空间，返回拷贝的字符串的大小

该函数会对字符串的地址空间进行校验，防止出现地址空间越界的问题。

参数

src

源字符串

dst

目标字符串

size

要拷贝的源字符串的长度

```
int strcmp(char *FirstPart, char *SecondPart)
```

描述

比较两个字符串的大小。

返回值

情况	返回值
FirstPart == SecondPart	0
FirstPart > SecondPart	1
FirstPart < SecondPart	-1

参数

FirstPart

第一个字符串

SecondPart

第二个字符串

```
printk(const char* fmt, ...)
```

描述

该宏能够在控制台上以黑底白字格式化输出字符串.

参数

fmt

源格式字符串

...

可变参数

```
printk_color(unsigned int FRcolor, unsigned int BKcolor, const char* fmt, ...)
```

描述

在控制台上以指定前景色和背景色格式化输出字符串.

参数

FRcolor

前景色

BKcolor

背景色

fmt

源格式字符串

...

可变参数

```
int vsprintf(char *buf, const char *fmt, va_list args)
```

描述

按照 fmt 格式化字符串，并将结果输出到 buf 中，返回写入 buf 的字符数量。

参数

buf

输出缓冲区

fmt

源格式字符串

args

可变参数列表

```
int sprintk(char *buf, const char *fmt, ...)
```

描述

按照 fmt 格式化字符串，并将结果输出到 buf 中，返回写入 buf 的字符数量。

参数

buf

输出缓冲区

fmt

源格式字符串

...

可变参数

内存操作

```
void *memcpy(void *dst, const void *src, uint64_t size)
```

描述

将内存从 src 处拷贝到 dst 处。

参数

dst

指向目标地址的指针

src

指向源地址的指针

size

待拷贝的数据大小

```
void *memmove(void *dst, const void *src, uint64_t size)
```

描述

与 `memcpy()` 类似，但是在源数据区域与目标数据区域之间存在重合时，该函数能防止数据被错误的覆盖。

参数

dst

指向目标地址的指针

src

指向源地址的指针

size

待拷贝的数据大小

6.2 原子变量

6.2.1 简介

DragonOS 实现了原子变量，类型为 atomic_t. 原子变量是基于具体体系结构的原子操作指令实现的。具体实现在 `kernel/common/atomic.h` 中。

6.2.2 API

请注意，以下 API 均为原子操作。

```
inline void atomic_add(atomic_t *ato, long val)
```

描述

原子变量增加指定值

参数

ato

原子变量对象

val

变量要增加的值

```
inline void atomic_sub(atomic_t *ato, long val)
```

描述

原子变量减去指定值

参数

ato

原子变量对象

val

变量要被减去的值

```
void atomic_inc(atomic_t *ato)
```

描述

原子变量自增 1

参数

ato

原子变量对象

```
void atomic_dec(atomic_t *ato)
```

描述

原子变量自减 1

参数

ato

原子变量对象

```
inline void atomic_set_mask(atomic_t *ato, long mask)
```

描述

将原子变量的值与 mask 变量进行 or 运算

参数

ato

原子变量对象

mask

与原子变量进行 or 运算的变量

```
inline void atomic_clear_mask(atomic_t *ato, long mask)
```

描述

将原子变量的值与 mask 变量进行 and 运算

参数

ato

原子变量对象

mask

与原子变量进行 and 运算的变量

6.3 类型转换库 API

内核提供了一些函数来帮助你在不同的类型之间进行转换。包括以下类型：

- 数值类型转换（使用 `num-traits` 库）
- Arc 类型转换

上述没有特殊标明的函数，都是在 `kernel/src/libs/casting.rs` 中实现的。

6.3.1 1. 数值类型转换

1.1. 整数类型与枚举类型之间的转换

您可以使用 `num-traits` 库提供的宏，实现枚举类型和整数类型之间的转换。SystemError 枚举类型使用了这种方式，您可以在 `kernel/src/syscall/mod.rs` 中找到它的用法。

它首先继承了 `FromPrimitive, ToPrimitive` 两个 trait，然后这样转换：

```
impl SystemError {
    /// @brief 把posix错误码转换为系统错误枚举类型。
    pub fn from_posix_errno(errno: i32) -> Option<SystemError> {
        // posix 错误码是小于0的
        if errno >= 0 {
            return None;
        }
        return <Self as FromPrimitive>::from_i32(-errno);
    }

    /// @brief 把系统错误枚举类型转换为负数posix错误码。
    pub fn to_posix_errno(&self) -> i32 {
        return -<Self as ToPrimitive>::to_i32(self).unwrap();
    }
}
```

这两个函数很好的说明了如何使用这两个 trait。

6.3.2 2. Arc 类型转换

2.1 从 Arc 转换为 Arc

当我们需要把一个 `Arc<dyn U>` 转换为 `Arc<T>` 的具体类型指针时，我们要为 `U` 这个 trait 实现 `DowncastArc`trait。这个 trait 定义在 `kernel/src/libs/casting.rs` 中。它要求 `trait U` 实现 `Any + Sync + Send`trait.

为 `trait U: Any + Send + Sync` 实现 `DowncastArc`trait，需要这样做：

```
impl DowncastArc for dyn U {
    fn as_any_arc(self: Arc<Self>) -> Arc<dyn Any> {
        return self;
    }
}
```

使用 `DowncastArc`trait，我们可以这样转换：

```
let arc: Arc<dyn U> = ...;
let arc_t: Arc<T> = arc.downcast_arc::<T>().unwrap();
```

如果 `arc` 的具体类型不是 `Arc<T>`，那么 `downcast_arc::<T>()` 会返回 `None`。

6.4 Notifier Chian 通知链

6.4.1 1. 原理概要

通知链是内核中各个子系统之间或子系统内部各个模块之间的事件通知机制。通知链本质上是一个事件处理函数的列表，每个通知链都与某个类型的事件相关（例如 reboot 事件）。当特定的事件发生时，就会调用相应事件的通知链中的回调函数，使子系统/模块对事件响应，并进行相对应的处理。

通知链和订阅功能有些类似，可以理解为：有个“通知者”维护了一个列表，“订阅者”将自己的回调函数注册进这个列表（“订阅者”当然也可以注销自己的回调函数）。当某个事件发生需要通知时，“通知者”就遍历这个列表中所有的回调函数并进行调用，这样所有注册的“订阅者”就能针对这个事件做出自己的响应和处理。

6.4.2 2. 核心功能

2.1 注册回调函数

将回调函数封装成特定的结构体，并将该结构体注册到指定的通知链当中。相关方法为 `register`，由“订阅者”使用。

2.2 注销回调函数

将回调函数从指定的通知链当中进行注销，即从通知链中删去该回调函数。相关方法为 `unregister`，由“订阅者”使用。

2.3 事件通知

当某个事件发生时，该事件相关的通知链通过该方法来进行事件通知。`call_chain` 这个方法会遍历通知链中的所有元素，并依次调用注册的回调函数。该方法由“通知者”使用。

6.4.3 3. 通知链类型

每种通知链都有相对应的 `register`，`unregister` 以及 `call_chain` 的接口，其功能同上面所述的核心功能。

- `AtomicNotifierChain`：原子的通知链，不可睡眠，建议用于中断上下文。
- `BlockingNotifierChain`：可阻塞的通知链，可以睡眠，建议用在进程上下文。
- `RawNotifierChain`：原始的通知链，由调用者自行考虑线程安全。

6.4.4 4. 其它问题

`BlockingNotifierChain` 暂时没实现可睡眠的功能。

6.5 软中断

软件中断，也可以被称为中断的下半部，用于延迟处理硬中断（中断上半部）未完成的工作。将中断分为两个阶段可以有效解决中断处理时间过长和中断丢失的问题。

6.5.1 1. 设计思路

每个 cpu 都有自己的 pending，软中断是“哪个 cpu 发起，就哪个 cpu 执行”，每个 cpu 的 pending 不共享。同一个软中断向量可以在多核上同时运行。

当我们需要注册一个新的软中断时，需要为软中断处理程序实现 `SoftirqVec` 特征，然后调用 `register_softirq` 函数，将软中断处理程序注册到软中断机制内。

请注意，由于软中断的可重入、可并发性，所以软中断处理程序需要自己保证线程安全。

6.5.2 2. 软中断向量号

```
pub enum SoftirqNumber {
    /// 时钟软中断信号
    TIMER = 0,
    /// 帧缓冲区刷新软中断
    VideoRefresh = 1,
}
```

6.5.3 3. 软中断 API

3.1. SoftirqVec 特征

```
pub trait SoftirqVec: Send + Sync + Debug {
    fn run(&self);
}
```

软中断处理程序需要实现的特征，需要实现 `run` 函数，用于处理软中断。当软中断被执行时，会调用 `run` 函数。

3.2. Softirq 的 API

3.2.1. 注册软中断向量

```
pub fn register_softirq(&self,
        softirq_num: SoftirqNumber,
        handler: Arc<dyn SoftirqVec>,
    ) -> Result<i32, SystemError>
```

- 参数：
  - softirq_num：中断向量号

  - hanlder：中断函数对应的结构体，需要指向实现了 `SoftirqVec` 特征的结构体变量

- 返回：
  - Ok(i32)：0
  - Err(SystemError)：错误码

3.2.2. 解注册软中断向量

```
pub fn unregister_softirq(&self, softirq_num: SoftirqNumber)
```

- 参数：
  - softirq_num：中断向量号

3.2.3. 软中断执行

```
pub fn do_softirq(&self)
```

- 作用：执行软中断函数（**只在硬中断执行后调用**）

3.2.4. 清除软中断的 pending 标志

```
pub unsafe fn clear_softirq_pending(&self, softirq_num: SoftirqNumber)
```

- 作用：清除当前 CPU 上，指定软中断的 pending 标志。请注意，这个函数是 unsafe 的，因为它会直接修改 pending 标志，而没有加锁。
- 参数：
  - softirq_num：中断向量号

3.2.5. 标志软中断需要执行

```
pub fn raise_softirq(&self, softirq_num: SoftirqNumber)
```

- 作用：标志当前 CPU 上，指定的软中断需要执行
- 参数：
  - softirq_num：中断向量号

3.3. 使用实例

```
#[derive(Debug)]
/// SoftirqExample中断结构体
pub struct SoftirqExample {
    running: AtomicBool,
}
/// SoftirqExample中断需要处理的逻辑
fn softirq_example_func() {
    println!("addressed SoftirqExample");
}
impl SoftirqVec for SoftirqExample {
    fn run(&self) {
        if self.set_run() == false {
            return;
        }

        softirq_example_func();

        self.clear_run();
    }
}
impl SoftirqExample {
    pub fn new() -> SoftirqExample {
        SoftirqExample {
            running: AtomicBool::new(false),
        }
    }

    fn set_run(&self) -> bool {
        let x = self
            .running
            .compare_exchange(false, true, Ordering::Acquire, Ordering::Relaxed);
        if x.is_ok() {
            return true;
        } else {
            return false;
        }
    }

    fn clear_run(&self) {
        self.running.store(false, Ordering::Release);
    }
}
```

(下页继续)

(续上页)

```
fn main() {
    let softirq_example = Arc::new(SoftirqExample::new());
    let softirq_num = 2;
    // 注册SoftirqExample中断
    softirq_vectors()
        .register_softirq(SoftirqNumber::from(softirq_num as u64), softirq_example)
        .expect("failed to register SoftirqExample");

    // 标志SoftirqExample中断需要执行
    softirq_vectors().raise_softirq(SoftirqNumber::from(softirq_num as u64));

    // 标志SoftirqExample中断不需要执行
    softirq_vectors().clear_softirq_pending(SoftirqNumber::from(softirq_num as u64));

    // 解注册SoftirqExample中断
    softirq_vectors().unregister_softirq(SoftirqNumber::from(softirq_num as u64));
}
```

CHAPTER 7

锁

这里是 DragonOS 的锁变量的说明文档。

7.1 锁的类型及其规则

7.1.1 简介

DragonOS 内核实现了一些锁，大致可以分为两类：

- 休眠锁
- 自旋锁

7.1.2 锁的类型

休眠锁

休眠锁只能在可抢占的上下文之中被获取。

在 DragonOS 之中，实现了以下的休眠锁：

- semaphore
- mutex_t

自旋锁

- spinlock_t
- *RawSpinLock*（Rust 版本的 spinlock_t，但与 spinlock_t 不兼容）
- *SpinLock* ——在 RawSpinLock 的基础上，封装了一层守卫 (Guard), 将锁及其要保护到的数据绑定在一个结构体内，并能在编译期避免未加锁就访问数据的问题。

进程在获取自旋锁后，将改变 pcb 中的锁变量持有计数，从而隐式地禁止了抢占。为了获得更多灵活的操作，spinlock 还提供了以下的方法：

后缀	说明
_irq()	在加锁时关闭中断/在放锁时开启中断
_irqsave()/_irqrestore()	在加锁时保存中断状态，并关中断/在放锁时恢复中断状态

7.1.3 详细介绍

自旋锁的详细介绍

关于自旋锁的详细介绍，请见文档：自旋锁

semaphore 信号量

semaphore 信号量是基于计数实现的。

当可用资源不足时，尝试对 semaphore 执行 down 操作的进程将会被休眠，直到资源可用。

mutex 互斥量

请见*Mutex 文档*

备注： 作者：龙进 longjin@RinGoTek.cn

7.2 自旋锁

7.2.1 1. 简介

自旋锁是用于多线程同步的一种锁，线程反复检查锁变量是否可用。由于线程在这一过程中保持运行的状态，因此是一种忙等待。一旦获取了自旋锁，线程会一直保持该锁，直至显式释放自旋锁。

DragonOS 在 kernel/src/lib/spinlock.rs 文件中，实现了自旋锁。根据功能特性的略微差异，分别提供了 RawSpinLock 和 SpinLock 两种自旋锁。

7.2.2 2. RawSpinLock - 原始自旋锁

RawSpinLock 是原始的自旋锁，其数据部分包含一个 AtomicBool, 实现了自旋锁的基本功能。其加锁、放锁需要手动确定对应的时机，也就是说，和我们在其他语言中使用的自旋锁一样，需要先调用 lock() 方法，然后当离开临界区时，手动调用 unlock() 方法。我们并没有向编译器显式地指定该自旋锁到底保护的是哪些数据。

RawSpinLock 为程序员提供了非常自由的加锁、放锁控制。但是，正是由于它过于自由，因此在使用它的时候，我们很容易出错。很容易出现“未加锁就访问临界区的数据”、“忘记放锁”、“双重释放”等问题。当使用 RawSpinLock 时，编译器并不能对这些情况进行检查，这些问题只能在运行时被发现。

警告： RawSpinLock 与 C 版本的 spinlock_t 不具有二进制兼容性。如果由于暂时的兼容性的需求，要操作 C 版本的 spinlock_t, 请使用 spinlock.rs 中提供的 C 版本的 spinlock_t 的操作函数。

但是，对于新开发的功能，请不要使用 C 版本的 spinlock_t，因为随着代码重构的进行，我们将会移除它。

7.2.3 3. SpinLock - 具备守卫的自旋锁

SpinLock 在 RawSpinLock 的基础上，进行了封装，能够在编译期检查出“未加锁就访问临界区的数据”、“忘记放锁”、“双重释放”等问题；并且，支持数据的内部可变性。

其结构体原型如下：

```
#[derive(Debug)]
pub struct SpinLock<T> {
    lock: RawSpinlock,
    /// 自旋锁保护的数据
    data: UnsafeCell<T>,
}
```

3.1. 使用方法

您可以这样初始化一个 SpinLock：

```
let x = SpinLock::new(Vec::new());
```

在初始化这个 SpinLock 时，必须把要保护的数据传入 SpinLock，由 SpinLock 进行管理。

当需要读取、修改 SpinLock 保护的数据时，请先使用 SpinLock 的 `lock()` 方法。该方法会返回一个 `SpinLockGuard`。您可以使用被保护的数据的成员函数来进行一些操作。或者是直接读取、写入被保护的数据。(相当于您获得了被保护的数据的可变引用)

完整示例如下方代码所示：

```
let x :SpinLock<Vec<i32>>= SpinLock::new(Vec::new());
    {
        let mut g :SpinLockGuard<Vec<i32>>= x.lock();
        g.push(1);
        g.push(2);
        g.push(2);
        assert!(g.as_slice() == [1, 2, 2] || g.as_slice() == [2, 2, 1]);
        // 在此处，SpinLock是加锁的状态
        kdebug!("x={:?}", x);
    }
    //␣
↪由于上方的变量`g`，也就是SpinLock守卫的生命周期结束，自动释放了SpinLock。因此，在此处，SpinLock是放
    kdebug!("x={:?}", x);
```

对于结构体内部的变量，我们可以使用 SpinLock 进行细粒度的加锁，也就是使用 SpinLock 包裹需要细致加锁的成员变量，比如这样：

```
pub struct a {
  pub data: SpinLock<data_struct>,
}
```

当然，我们也可以对整个结构体进行加锁：

```
struct MyStruct {
  pub data: data_struct,
}
/// 被全局加锁的结构体
pub struct LockedMyStruct(SpinLock<MyStruct>);
```

3.2. 原理

SpinLock 之所以能够实现编译期检查，是因为它引入了一个 SpinLockGuard 作为守卫。我们在编写代码的时候，保证只有调用 SpinLock 的 `lock()` 方法加锁后，才能生成一个 SpinLockGuard。并且，当我们想要访问受保护的数据的时候，都必须获得一个守卫。然后，我们为 SpinLockGuard 实现了 Drop trait，当守卫的生命周期结束时，将会自动释放锁。除此以外，没有别的方法能够释放锁。因此我们能够得知，一个上下文中，只要 SpinLockGuard 的生命周期没有结束，那么它就拥有临界区数据的访问权，数据访问就是安全的。

3.3. 存在的问题

3.3.1. 双重加锁

请注意，SpinLock 支持的编译期检查并不是万能的。它目前无法在编译期检查出“双重加锁”问题。试看这样一个场景：函数 A 中，获得了锁。然后函数 B 中继续尝试加锁，那么就造成了“双重加锁”问题。这样在编译期是无法检测出来的。

针对这个问题，我们建议采用这样的编程方法：

- 如果函数 B 需要访问临界区内的数据，那么，函数 B 应当接收一个类型为 &SpinLockGuard 的参数，这个守卫由函数 A 获得。这样一来，函数 B 就能访问临界区内的数据。

备注： 作者：龙进 longjin@RinGoTek.cn

7.3 mutex 互斥量

mutex 是一种轻量级的同步原语，只有被加锁、空闲两种状态。

当 mutex 被占用时，尝试对 mutex 进行加锁操作的进程将会被休眠，直到资源可用。

7.3.1 1. 特性

- 同一时间只有 1 个任务可以持有 mutex
- 不允许递归地加锁、解锁
- 只允许通过 mutex 的 api 来操作 mutex
- 在硬中断、软中断中不能使用 mutex

7.3.2 2. 定义

mutex 定义在 lib/mutex.rs 中，定义如下所示：

```
/// @brief Mutex互斥量结构体
///␣
↪请注意！由于Mutex属于休眠锁，因此，如果您的代码可能在中断上下文内执行，请勿采用Mutex!
#[derive(Debug)]
pub struct Mutex<T> {
    /// 该Mutex保护的数据
    data: UnsafeCell<T>,
    /// Mutex内部的信息
```

(下页继续)

(续上页)

```
    inner: SpinLock<MutexInner>,
}

#[derive(Debug)]
struct MutexInner {
    /// 当前Mutex是否已经被上锁(上锁时，为true)
    is_locked: bool,
    /// 等待获得这个锁的进程的链表
    wait_list: LinkedList<&'static mut process_control_block>,
}
```

7.3.3 3. 使用

与 SpinLock 类似，Rust 版本的 Mutex 具有一个守卫。使用的时候，需要将要被保护的数据的所有权移交 Mutex。并且，守卫只能在加锁成功后产生，因此，每个时刻，每个 Mutex 最多存在 1 个守卫。

当需要读取、修改 Mutex 保护的数据时，请先使用 Mutex 的 `lock()` 方法。该方法会返回一个 `MutexGuard`。您可以使用被保护的数据的成员函数来进行一些操作。或者是直接读取、写入被保护的数据。(相当于您获得了被保护的数据的可变引用)

完整示例如下方代码所示：

```
let x :Mutex<Vec<i32>>= Mutex::new(Vec::new());
    {
        let mut g :MutexGuard<Vec<i32>>= x.lock();
        g.push(1);
        g.push(2);
        g.push(2);
        assert!(g.as_slice() == [1, 2, 2] || g.as_slice() == [2, 2, 1]);
        // 在此处，Mutex是加锁的状态
        kdebug!("x={:?}", x);
    }
    //␣
↪由于上方的变量`g`，也就是Mutex守卫的生命周期结束，自动释放了Mutex。因此，在此处，Mutex是放锁的状态
    kdebug!("x={:?}", x);
```

对于结构体内部的变量，我们可以使用 Mutex 进行细粒度的加锁，也就是使用 Mutex 包裹需要细致加锁的成员变量，比如这样：

```
pub struct a {
  pub data: Mutex<data_struct>,
}
```

当然，我们也可以对整个结构体进行加锁：

```
struct MyStruct {
  pub data: data_struct,
}
/// 被全局加锁的结构体
pub struct LockedMyStruct(Mutex<MyStruct>);
```

7.3.4 4. API

4.1. new - 初始化 Mutex

原型

```
pub const fn new(value: T) -> Self
```

说明

new() 方法用于初始化一个 Mutex。该方法需要一个被保护的数据作为参数。并且，该方法会返回一个 Mutex。

4.2. lock - 加锁

原型

```
pub fn lock(&self) -> MutexGuard<T>
```

说明

对 Mutex 加锁，返回 Mutex 的守卫，您可以使用这个守卫来操作被保护的数据。

如果 Mutex 已经被加锁，那么，该方法会阻塞当前进程，直到 Mutex 被释放。

4.3. try_lock - 尝试加锁

原型

```
pub fn try_lock(&self) -> Result<MutexGuard<T>, i32>
```

说明

尝试对 Mutex 加锁。如果加锁失败，不会将当前进程加入等待队列。如果加锁成功，返回 Mutex 的守卫；如果当前 Mutex 已经被加锁，返回 `Err(错误码)`。

7.3.5 5. C 版本的 Mutex（在将来会被废弃）

mutex 定义在 `common/mutex.h` 中。其数据类型如下所示：

```
typedef struct
{

    atomic_t count; // 锁计数。1->已解锁。 0->已上锁,且有可能存在等待者
    spinlock_t wait_lock;   // mutex操作锁，用于对mutex的list的操作进行加锁
    struct List wait_list;  // Mutex的等待队列
} mutex_t;
```

5.1. API

mutex_init

void mutex_init(mutex_t *lock)

初始化一个 mutex 对象。

mutex_lock

void mutex_lock(mutex_t *lock)

对一个 mutex 对象加锁。若 mutex 当前被其他进程持有，则当前进程进入休眠状态。

mutex_unlock

void mutex_unlock(mutex_t *lock)

对一个 mutex 对象解锁。若 mutex 的等待队列中有其他的进程，则唤醒下一个进程。

mutex_trylock

void mutex_trylock(mutex_t *lock)

尝试对一个 mutex 对象加锁。若 mutex 当前被其他进程持有，则返回 0. 否则，加锁成功，返回 1.

mutex_is_locked

void mutex_is_locked(mutex_t *lock)

判断 mutex 是否已被加锁。若给定的 mutex 已处于上锁状态，则返回 1，否则返回 0。

7.4 RwLock 读写锁

备注： 本文作者: sujintao

Email: sujintao@dragonos.org

7.4.1 1. 简介

读写锁是一种在并发环境下保护多进程间共享数据的机制. 相比于普通的 spinlock, 读写锁将对共享数据的访问分为读和写两种类型: 只读取共享数据的访问使用读锁控制, 修改共享数据的访问使用写锁控制. 读写锁设计允许同时存在多个”读者”(只读取共享数据的访问) 和一个”写者”(修改共享数据的访问), 对于一些大部分情况都是读访问的共享数据来说, 使用读写锁控制访问可以一定程度上提升性能.

7.4.2 2. DragonOS 中读写锁的实现

2.1 读写锁的机理

读写锁的目的是维护多线程系统中的共享变量的一致性. 数据会被包裹在一个 RwLock 的数据结构中, 一切的访问必须通过 RwLock 的数据结构进行访问和修改. 每个要访问共享数据的会获得一个守卫 (guard), 只读进程获得 READER(读者守卫), 需要修改共享变量的进程获得 WRITER(写者守卫), 作为 RwLock 的”影子”, 线程都根据 guard 来进行访问和修改操作.

在实践中, 读写锁除了 READER, WRITER, 还增加了 UPGRADER; 这是一种介于 READER 和 WRITER 之间的守卫, 这个守卫的作用就是防止 WRITER 的饿死 (Staration). 当进程获得 UPGRADER 时, 进程把它当成 READER 来使用; 但是 UPGRADER 可以进行升级处理, 升级后的 UPGRADER 相当于是一个 WRITER 守卫, 可以对共享数据执行写操作.

所有守卫都满足 rust 原生的 RAII 机理, 当守卫所在的作用域结束时, 守卫将自动释放.

2.2 读写锁守卫之间的关系

同一时间点, 可以存在多个 READER, 即可以同时有多个进程对共享数据进行访问; 同一时间只能存在一个 WRITER, 而且当有一个进程获得 WRITER 时, 不能存在 READER 和 UPGRADER; 进程获得 UPGRADER 的前提条件是, 不能有 UPGRADER 或 WRITER 存在, 但是当有一个进程获得 UPGRADER 时, 进程无法成功申请 READER.

2.3 设计的细节

2.3.1 RwLock 数据结构

```
pub struct RwLock<T> {
    lock: AtomicU32,//原子变量
    data: UnsafeCell<T>,
}
```

2.3.2 READER 守卫的数据结构

```
pub struct RwLockReadGuard<'a, T: 'a> {
    data: *const T,
    lock: &'a AtomicU32,
}
```

2.3.3 UPGRADER 守卫的数据结构

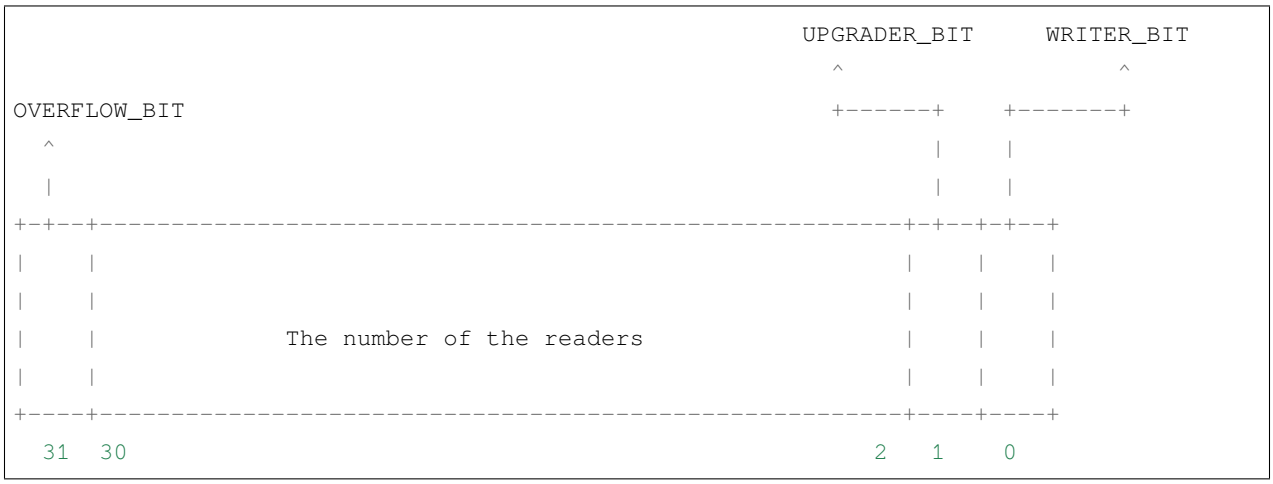
```
pub struct RwLockUpgradableGuard<'a, T: 'a> {
    data: *const T,
    inner: &'a RwLock<T>,
}
```

2.3.4 WRITER 守卫的数据结构

```
pub struct RwLockWriteGuard<'a, T: 'a> {
    data: *mut T,
    inner: &'a RwLock<T>,
}
```

2.3.5 RwLock 的 lock 的结构介绍

lock 是一个 32 位原子变量 AtomicU32, 它的比特位分配如下:



(从右到左) 第 0 位表征 WRITER 是否有效, 若 WRITER_BIT=1, 则存在一个进程获得了 WRITER 守卫; 若 UPGRADER_BIT=1, 则存在一个进程获得了 UPGRADER 守卫, 第 2 位到第 30 位用来二进制表示获得 READER 守卫的进程数; 第 31 位是溢出判断位, 若 OVERFLOW_BIT=1, 则不再接受新的读者守卫的获得申请.

7.4.3 3. 读写锁的主要 API

3.1 RwLock 的主要 API

```
///功能:  输入需要保护的数据类型data,返回一个新的RwLock类型.
pub const fn new(data: T) -> Self
```

```
///功能: 获得READER守卫
pub fn read(&self) -> RwLockReadGuard<T>
```

```
///功能: 尝试获得READER守卫
pub fn try_read(&self) -> Option<RwLockReadGuard<T>>
```

```
///功能: 获得WRITER守卫
pub fn write(&self) -> RwLockWriteGuard<T>
```

```
///功能: 尝试获得WRITER守卫
pub fn try_write(&self) -> Option<RwLockWriteGuard<T>>
```

```
///功能：获得UPGRADER守卫
pub fn upgradeable_read(&self) -> RwLockUpgradableGuard<T>
```

```
///功能：尝试获得UPGRADER守卫
pub fn try_upgradeable_read(&self) -> Option<RwLockUpgradableGuard<T>>
```

3.2 WRITER 守卫 RwLockWriteGuard 的主要 API

```
///功能：将WRITER降级为READER
pub fn downgrade(self) -> RwLockReadGuard<'rwlock, T>
```

```
///功能：将WRITER降级为UPGRADER
pub fn downgrade_to_upgradeable(self) -> RwLockUpgradableGuard<'rwlock, T>
```

3.3 UPGRADER 守卫 RwLockUpgradableGuard 的主要 API

```
///功能：将UPGRADER升级为WRITER
pub fn upgrade(mut self) -> RwLockWriteGuard<'rwlock, T>
```

```
///功能：将UPGRADER降级为READER
pub fn downgrade(self) -> RwLockReadGuard<'rwlock, T>
```

7.4.4 4. 用法实例

```
static LOCK: RwLock<u32> = RwLock::new(100 as u32);

fn t_read1() {
    let guard = LOCK.read();
    let value = *guard;
    let readers_current = LOCK.reader_count();
    let writers_current = LOCK.writer_count();
    println!(
        "Reader1: the value is {value}
    There are totally {writers_current} writers, {readers_current} readers"
    );
}

fn t_read2() {
    let guard = LOCK.read();
```

(下页继续)

(续上页)

```
    let value = *guard;
    let readers_current = LOCK.reader_count();
    let writers_current = LOCK.writer_count();
    println!(
        "Reader2: the value is {value}
    There are totally {writers_current} writers, {readers_current} readers"
    );
}

fn t_write() {
    let mut guard = LOCK.write();
    *guard += 100;
    let writers_current = LOCK.writer_count();
    let readers_current = LOCK.reader_count();
    println!(
        "Writers: the value is {guard}
    There are totally {writers_current} writers, {readers_current} readers",
        guard = *guard
    );
    let read_guard=guard.downgrade();
    let value=*read_guard;
    println!("After downgraded to read_guard: {value}");
}

fn t_upgrade() {
    let guard = LOCK.upgradeable_read();
    let value = *guard;
    let readers_current = LOCK.reader_count();
    let writers_current = LOCK.writer_count();
    println!(
        "Upgrader1 before upgrade: the value is {value}
    There are totally {writers_current} writers, {readers_current} readers"
    );
    let mut upgraded_guard = guard.upgrade();
    *upgraded_guard += 100;
    let writers_current = LOCK.writer_count();
    let readers_current = LOCK.reader_count();
    println!(
        "Upgrader1 after upgrade: the value is {temp}
    There are totally {writers_current} writers, {readers_current} readers",
        temp = *upgraded_guard
    );
    let downgraded_guard=upgraded_guard.downgrade_to_upgradeable();
```

(下页继续)

(续上页)

```
    let value=*downgraded_guard;
    println!("value after downgraded: {value}");
    let read_guard=downgraded_guard.downgrade();
    let value_=*read_guard;
    println!("value after downgraded to read_guard: {value_}");
}

fn main() {
    let r2=thread::spawn(t_read2);
    let r1 = thread::spawn(t_read1);
    let t1 = thread::spawn(t_write);
    let g1 = thread::spawn(t_upgrade);
    r1.join().expect("r1");
    t1.join().expect("t1");
    g1.join().expect("g1");
    r2.join().expect("r2");
}
```

CHAPTER 8

进程管理模块

8.1 kthread 内核线程

内核线程模块实现在 process/kthread.rs 中，提供对内核线程的及支持功能。内核线程作为内核的“分身”，能够提升系统的并行化程度以及故障容错能力。

8.1.1 原理

每个内核线程都运行在内核态，执行其特定的任务。

内核线程的创建是通过调用 KernelThreadMechanism::create() 或者 KernelThreadMechanism::create_and_run() 函数，向 kthreadd 守护线程发送创建任务来实现的。也就是说，内核线程的创建，最终是由 kthread_daemon 来完成。

当内核线程被创建后，默认处于睡眠状态，要使用 ProcessManager::wakeup 函数将其唤醒。当内核其他模块想要停止一个内核线程的时候，可以调用 KernelThreadMechanism::stop() 函数，等待内核线程的退出，然后获得返回值并清理内核线程的 pcb。

内核线程应当经常检查 KernelThreadMechanism::should_stop() 的结果，以确定其是否要退出。当检测到需要退出时，内核线程返回一个返回码，即可退出。(注意资源的清理)

8.2 加载程序

8.2.1 1. 二进制程序装载

在小节，你将了解 DragonOS 的二进制程序加载器的原理。

DragonOS 在装载二进制程序时，执行了“探测-装载”的过程。

在探测阶段，DragonOS 会读取文件首部，然后依次调用各个二进制加载器的探测函数，判断该二进制程序是否适用于该加载器。如果适用，则使用这个加载器进行装载。

在装载阶段，DragonOS 会使用上述加载器进行装载。装载器会将二进制程序的各个段映射到内存中，并且得到二进制程序的入口地址。

备注： 目前 DragonOS 不支持动态链接，因此所有的二进制程序都是静态链接的。并且暂时支持的只有 ELF 加载器。

CHAPTER 9

DragonOS 调度

这里是 DragonOS 中，与进程调度相关的说明文档。

9.1 与“等待”相关的 api（C 语言）

警告： 随着内核的发展，我们将会逐步将 C 语言的等待机制替换为 Rust 语言的等待机制。在这个过程中，我们将会同时保留 C 语言和 Rust 语言的等待机制，以便于我们在开发过程中进行对比。待时机成熟，我们将会逐步将 C 语言的等待机制移除。

如果几个进程需要等待某个事件发生，才能被运行，那么就需要一种“等待”的机制，以实现进程同步。

9.1.1 一. wait_queue 等待队列

wait_queue 是一种进程同步机制，中文名为“等待队列”。它可以将当前进程挂起，并在时机成熟时，由另一个进程唤醒他们。

当您需要等待一个事件完成时，使用 wait_queue 机制能减少进程同步的开销。相比于滥用自旋锁以及信号量，或者是循环使用 usleep(1000) 这样的函数来完成同步，wait_queue 是一个高效的解决方案。

警告： `wait_queue.h` 中的等待队列的实现并没有把队列头独立出来，同时没有考虑为等待队列加锁。所以在后来的开发中加入了 `wait_queue_head_t` 的队列头实现，实质上就是链表 + 自旋锁。它与

> wait_queue.h 中的队列 wait_queue_node_t 是兼容的，当你使用 struct wait_queue_head 作为队列头时，你同样可以使用等待队列添加节点的函数。

简单用法

等待队列的使用方法主要分为以下几部分：

- 创建并初始化一个等待队列
- 使用 wait_queue_sleep_on_ 系列的函数，将当前进程挂起。晚挂起的进程将排在队列的尾部。
- 通过 wait_queue_wakeup() 函数，依次唤醒在等待队列上的进程，将其加入调度队列

要使用 wait_queue，您需要 #include<common/wait_queue.h>，并创建一个 wait_queue_node_t 类型的变量，作为等待队列的头部。这个结构体只包含两个成员变量：

```
typedef struct
{
    struct List wait_list;
    struct process_control_block *pcb;
} wait_queue_node_t;
```

对于等待队列，这里有一个好的命名方法：

```
wait_queue_node_t wq_keyboard_interrupt_received;
```

这样的命名方式能增加代码的可读性，更容易让人明白这里到底在等待什么。

初始化等待队列

函数 wait_queue_init(wait_queue_node_t *wait_queue, struct process_control_block *pcb) 提供了初始化 wait_queue 结点的功能。

当您初始化队列头部时，您仅需要将 wait_queue 首部的结点指针传入，第二个参数请设置为 NULL

将结点插入等待队列

您可以使用以下函数，将当前进程挂起，并插入到指定的等待队列。这些函数大体功能相同，只是在一些细节上有所不同。

函数名	解释
wait_queue_sleep_on()	将当前进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE
wait_queue_sleep_on_unlock()	将当前进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的自旋锁
wait_queue_sleep_on_interruptible()	将当前进程挂起，并设置挂起状态为 PROC_INTERRUPTIBLE

从等待队列唤醒一个进程

您可以使用 `void wait_queue_wakeup(wait_queue_node_t * wait_queue_head, int64_t state);` 函数，从指定的等待队列中，唤醒第一个挂起时的状态与指定的 `state` 相同的进程。

当没有符合条件的进程时，将不会唤醒任何进程，仿佛无事发生。

9.1.2 二. wait_queue_head 等待队列头

数据结构定义如下：

```
typedef struct
{
    struct List wait_list;
    spinlock_t lock;  // 队列需要有一个自旋锁,虽然目前内部并没有使用,但是以后可能会用.
} wait_queue_head_t;
```

等待队列头的使用逻辑与等待队列实际是一样的，因为他同样也是等待队列的节点 (仅仅多了一把锁)。且 wait_queue_head 的函数基本上与 wait_queue 一致，只不过多了 ***_with_node_*** 的字符串。

同时，wait_queue.h 文件中提供了很多的宏，可以方便您的工作。

提供的宏

宏	解释
DECLARE_WAIT_ON_STACK(name, pcb)	在栈上声明一个 wait_queue 节点，同时把 pcb 所代表的进程与该节点绑定
DECLARE_WAIT_ON_STACK_SELF(name)	传在栈上声明一个 wait_queue 节点，同时当前进程 (即自身进程) 与该节点绑定
DECLARE_WAIT_ALLOC(name, pcb)	使用 `kzalloc` 声明一个 wait_queue 节点，同时把 pcb 所代表的进程与该节点绑定，请记得使用 kfree 释放空间
DECLARE_WAIT_ALLOC_SELF(name)	使用 `kzalloc` 声明一个 wait_queue 节点，同时当前进程 (即自身进程) 与该节点绑定，请记得使用 kfree 释放空间

创建等待队列头

您可以直接调用宏

```
DECLARE_WAIT_QUEUE_HEAD(m_wait_queue_head);  // 在栈上声明一个队列头变量
```

也可以手动声明

```
struct wait_queue_head_t m_wait_queue_head = {0};
wait_queue_head_init(&m_wait_queue_head);
```

将结点插入等待队列

函数名	解释
wait_queue_sleep_with_node(wait_queue_head_t *head, wait_queue_node_t *wait_node)	传入一个等待队列节点，并设置该节点的挂起状态为 PROC_UNINTERRUPTIBLE
wait_queue_sleep_with_node_unlock(wait_queue_head_t *q, wait_queue_node_t *wait, void *lock)	传入一个等待队列节点，将该节点的 pcb 指向的进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的自旋锁
wait_queue_sleep_with_node_interriptible(wait_queue_head_t *q, wait_queue_node_t *wait)	传入一个等待队列节点，将该节点的 pcb 指向的进程挂起，并设置挂起状态为 PROC_INTERRUPTIBLE

从等待队列唤醒一个进程

在 wait_queue.h 中的 wait_queue_wakeup 函数直接 kfree 掉了 wait_node 节点。对于在栈上的 wait_node, 您可以选择 wait_queue_wakeup_on_stack(wait_queue_head_t *q, int64_t state) 来唤醒队列里面的队列头节点。

9.1.3 三. completion 完成量

简单用法

完成量的使用方法主要分为以下几部分：

- 声明一个完成量 (可以在栈中/使用 kmalloc/使用数组)
- 使用 wait_for_completion 等待事件完成
- 使用 complete 唤醒等待的进程

等待操作

```
void wait_fun() {
    DECLARE_COMPLETION_ON_STACK(comp);  // 声明一个completion

    // .... do somethind here
    // 大部分情况是你使用kthread_run()创建了另一个线程
    // 你需要把comp变量传给这个线程，然后当前线程就会等待他的完成

    if (!try_wait_for_completion(&comp))  // 进入等待
        wait_for_completion(&comp);
}
```

完成操作

```
void kthread_fun(struct completion *comp) {
    // ..... 做一些事  .......
    // 这里你确定你完成了目标事件

    complete(&comp);
    // 或者你使用complete_all
    complete_all(&comp);
}
```

更多用法

kernel/sched/completion.c 文件夹中, 你可以看到 __test 开头的几个函数, 他们是 completion 模块的测试代码, 基本覆盖了 completion 的大部分函数. 你可以在这里查询函数使用方法.

初始化完成量

函数 `completion_init(struct completion *x)` 提供了初始化 completion 的功能。当你使用 `DECLARE_COMPLETION_ON_STACK` 来创建 (在栈上创建) 的时候, 会自动初始化.

关于完成量的 wait 系列函数

函数名	解释
wait_for_completion(struct completion *x)	将当前进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE。
wait_for_completion_timeout(struct completion *x, long timeout)	将当前进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE。当等待 timeout 时间 (jiffies 时间片) 之后, 自动唤醒进程。
wait_for_completion_interruptible(struct completion *x)	将当前进程挂起，并设置挂起状态为 PROC_INTERRUPTIBLE。
wait_for_completion_interruptible_timeout(struct completion *x, long timeout)	将当前进程挂起，并设置挂起状态为 PROC_INTERRUPTIBLE。当等待 timeout 时间 (jiffies 时间片) 之后, 自动唤醒进程。
wait_for_multicompletion(struct completion x[], int n)	将当前进程挂起，并设置挂起状态为 PROC_UNINTERRUPTIBLE。(等待数组里面的 completion 的完成)

关于完成量的 complete 系列函数

函数名	解释
complete(struct completion *x)	表明一个事件被完成, 从等待队列中唤醒一个进程
complete_all(struct completion *x)	表明与该 completion 有关的事件被标记为永久完成, 并唤醒等待队列中的所有进程

其他用于查询信息的函数

函数名	解释
completion_done(struct completion *x)	查询 completion 的 done 变量是不是大于 0，如果大于 0，返回 true；否则返回 false。在等待前加上这个函数有可能加速？(暂未经过实验测试，有待证明)
try_wait_for_completion(struct completion *x)	查询 completion 的 done 变量是不是大于 0，如果大于 0，返回 true(同时令 done-=1)；否则返回 false。在等待前加上这个函数有可能加速？（该函数和 `completion_done` 代码逻辑基本一致，但是会主动令 completion 的 done 变量减 1）

9.2 与“等待”相关的 api（rust 语言）

如果几个进程需要等待某个事件发生，才能被运行，那么就需要一种“等待”的机制，以实现进程同步。

9.2.1 1. WaitQueue 等待队列

WaitQueue 是一种进程同步机制，中文名为“等待队列”。它可以将当前进程挂起，并在时机成熟时，由另一个进程唤醒他们。

当您需要等待一个事件完成时，使用 WaitQueue 机制能减少进程同步的开销。相比于滥用自旋锁以及信号量，或者是循环使用 usleep(1000) 这样的函数来完成同步，WaitQueue 是一个高效的解决方案。

1.1 WaitQueue 的使用

WaitQueue 的使用非常简单，只需要三步：

1. 初始化一个 WaitQueue 对象。
2. 调用这个 WaitQueue 的挂起相关的 API，将当前进程挂起。
3. 当事件发生时，由另一个进程，调用这个 WaitQueue 的唤醒相关的 API，唤醒一个进程。

下面是一个简单的例子：

1.1.1 初始化一个 WaitQueue 对象

WaitQueue 对象的初始化非常简单，只需要调用 WaitQueue::INIT 即可。

```
let mut wq = WaitQueue::INIT;
```

1.1.2 挂起进程

您可以这样挂起当前进程：

```
wq.sleep();
```

当前进程会被挂起，直到有另一个进程调用了 `wq.wakeup()`。

1.1.3 唤醒进程

您可以这样唤醒一个进程：

```
// 唤醒等待队列头部的进程（如果它的state & PROC_INTERRUPTIBLE 不为0）
wq.wakeup(PROC_INTERRUPTIBLE);

// 唤醒等待队列头部的进程（如果它的state & PROC_UNINTERRUPTIBLE 不为0）
wq.wakeup(PROC_UNINTERRUPTIBLE);

// 唤醒等待队列头部的进程（无论它的state是什么）
wq.wakeup((-1) as u64);
```

1.2 API

1.2.1 挂起进程

您可以使用以下函数，将当前进程挂起，并插入到指定的等待队列。这些函数大体功能相同，只是在一些细节上有所不同。

函数名	解释
sleep()	将当前进程挂起，并设置进程状态为 PROC_INTERRUPTIBLE
sleep_uninterruptible()	将当前进程挂起，并设置进程状态为 PROC_UNINTERRUPTIBLE
sleep_unlock_spinlock()	将当前进程挂起，并设置进程状态为 PROC_INTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的自旋锁
sleep_unlock_mutex()	将当前进程挂起，并设置进程状态为 PROC_INTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的 Mutex
sleep_uninterruptible_unlock_spinlock()	将当前进程挂起，并设置进程状态为 PROC_UNINTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的自旋锁
sleep_uninterruptible_unlock_mutex()	将当前进程挂起，并设置进程状态为 PROC_UNINTERRUPTIBLE。待当前进程被插入等待队列后，解锁给定的 Mutex

1.2.2 唤醒进程

您可以使用 `wakeup(state)` 函数，唤醒等待队列中的第一个进程。如果这个进程的 state 与给定的 state 进行 and 操作之后，结果不为 0, 则唤醒它。

返回值：如果有进程被唤醒，则返回 true，否则返回 false。

1.2.3 其它 API

函数名	解释
len()	返回等待队列中的进程数量

9.3 进程调度器相关的 api

定义了 DragonOS 的进程调度相关的 api，是系统进行进程调度的接口。同时也抽象出了 Scheduler 的 trait，以供具体的调度器实现

9.3.1 1. 调度器介绍

一般来说，一个系统会同时处理多个请求，但是其资源是优先的，调度就是用来协调每个请求对资源的使用的方法。

1.1 主要函数

1. cpu_executing(): 获取指定的 cpu 上正在执行的进程的 pcb
2. sched_enqueue(): 将进程加入调度队列
3. sched_init(): 初始化进程调度器模块
4. sched_update_jiffies(): 当时钟中断到达时，更新时间片。*请注意，该函数只能被时钟中断处理程序调用*
5. sys_sched(): 让系统立即运行调度器的系统调用。*请注意，该系统调用不能由 ring3 的程序发起*

9.4 完全公平调度器相关的 api

CFS（Completely Fair Scheduler），顾名思义，完全公平调度器。CFS 作为主线调度器之一，也是最典型的 O(1) 调度器之一

9.4.1 1. CFSQueue 介绍

CFSQueue 是用来存放普通进程的调度队列，每个 CPU 维护一个 CFSQueue，主要使用 Vec 作为主要存储结构来实现。

1.1 主要函数

1. enqueue(): 将 pcb 入队列
2. dequeue(): 将 pcb 从调度队列中弹出, 若队列为空，则返回 IDLE 进程的 pcb
3. sort(): 将进程按照虚拟运行时间的升序进行排列

9.4.2 2. SchedulerCFS 介绍

CFS 调度器类，主要实现了 CFS 调度器类的初始化以及调度功能函数。

2.1 主要函数

1. sched(): 是对于 Scheduler trait 的 sched() 实现，是普通进程进行调度时的逻辑处理，该函数会返回接下来要执行的 pcb，若没有符合要求的 pcb，返回 None
2. enqueue(): 同样是对于 Scheduler trait 的 sched() 实现，将一个 pcb 加入调度器的调度队列
3. update_cpu_exec_proc_jiffies(): 更新这个 cpu 上，这个进程的可执行时间。
4. timer_update_jiffies(): 时钟中断到来时，由 sched 的 core 模块中的函数，调用本函数，更新 CFS 进程的可执行时间

9.5 实时进程调度器相关的 api

RT（realtime scheduler），实时调度器。实时调度是为了完成实时处理任务而分配 CPU 的调度方法。

DragonOS 的进程分为“实时进程”和“普通进程”两类；实时进程的优先级高于普通进程，如果当前的系统的执行队列中有“实时进程”，RT 调度器会优先选择实时进程；如果队列中会有多个实时进程，调度器会选择优先级最高的实时进程来执行；

9.5.1 1. RTQueue 介绍

RTQueue 是用来存放 state 为 running 的实时进程的调度队列，每个 CPU 维护一个 RTQueue，主要使用 Vec 作为主要存储结构来实现。

1.1 主要函数

1. enqueue(): 将 pcb 入队列
2. dequeue(): 将 pcb 出队列

9.5.2 2. SchedulerRT 介绍

RT 调度器类，主要实现了 RT 调度器类的初始化以及调度功能函数。

2.1 主要函数

1. pick_next_task_rt(): 获取当前 CPU 中的第一个需要执行的 RT pcb
2. sched(): 是对于 Scheduler trait 的 sched() 实现，是实时进程进行调度时的逻辑处理，该函数会返回接下来要执行的 pcb，若没有符合要求的 pcb，返回 None
3. enqueue(): 同样是对于 Scheduler trait 的 sched() 实现，将一个 pcb 加入调度器的调度队列

2.2 内核调度策略

目前在 DragonOS 中，主要的调度策略有 SCHED_NORMAL 策略 | SCHED_FIFO 策略 | SCHED_RT 策略，具体的调度策略为：

1. SCHED_NORMAL 策略：SCHED_NORMAL 是“绝对公平调度策略”，该策略的进程使用 CFS 进行调度。
2. SCHED_FIFO 策略：SCHED_FIFO 是“实时进程调度策略”，这是一种先进先出的调度策略，该策略不涉及到 CPU 时间片机制，在没有更高优先级进程的前提下，只能等待其他进程主动释放 CPU 资源；在 SCHED_FIFO 策略中，被调度器调度运行的进程，其运行时长不受限制，可以运行任意长的时间。
3. SCHED_RR 策略：SCHED_RR 是“实时进程调度策略”，使用的是时间片轮转机制，对应进程的 time_slice 会在运行时减少，进程使用完 CPU 时间片后，会加入该 CPU 的与该进程优先级相同的执行队列中。同时，释放 CPU 资源，CPU 的使用权会被分配给下一个执行的进程

9.5.3 3. Q&A

几种常用的方法

1. 如何创建实时进程

```
struct process_control_block *pcb_name = kthread_run_rt(&fn_name, NULL, "test␣
↪create rt pcb");
```

其中 kthread_run_rt，是创建内核实时线程的宏

2. pcb 中涉及到实时进程的字段含义
    1. policy：实时进程的策略，目前有：SCHED_FIFO 与 SCHED_RR
    2. priority: 实时进程的优先级，范围为 0-99，数字越大，表示优先级越高
    3. rt_time_slice: 实时进程的时间片，默认为 100，随着 CPU 运行而减少，在 rt_time_slice 为 0 时，将时间片赋初值并将该进程加入执行队列。
3. 如何实时进程存储队列
    - 目前是使用 Vec 来保存，因为具体实现的逻辑原因，目前的入队列和出队列都是对队尾的操作，因此会有如下现象：系统中有多个优先级相同的实时进程等待运行时，会出现饥饿现象，也即上一个因为时间片耗尽的进程会在下一个执行，造成同优先级等待的进程饥饿。
4. todo
    1. 将存储实时进程的队列使用双向链表存储（或者其他办法解决上述的饥饿问题）
    2. 目前的实时调度是针对单 CPU 的，需要实现多 CPU 的实时调度
    3. 实现 RT 进程和普通进程之间的分配带宽的比例
    4. 多个 CPU 之间实现负载均衡

9.6 内核定时器

9.6.1 1. 简介

内核定时器是内核中的一种定时器，内核定时器的工作方式是：添加定时器到队列，为每个定时器设置到期时间。当定时器到期时，会执行定时器对应的函数。

9.6.2 2. 设计思路

定时器类型为 Timer 结构体，而 Timer 由 SpinLock<InnerTimer> 组成。全局中使用元素类型为 Arc<Timer> 的队列 TIMER_LIST 存储系统创建的定时器。创建定时器时，应调用 Timer::new(timer_func,expire_jiffies)，timer_func 为定时器要执行的操作，expire_jiffies 为定时器的结束时间，timer_func 参数的类型是实现了 TimerFunction 特性的结构体。在创建定时器后，应使用 Timer::activate() 将定时器插入到 TIMER_LIST 中。

如果只是希望当前 pcb 休眠一段时间，应调用 schedule_timeout(timeout)，timeout 指定 pcb 休眠的时间长度。

9.6.3 3. 定时器应实现的特性

定时器要执行的函数应实现 TimerFunction 特性，其定义如下：

```
/// 定时器要执行的函数的特征
pub trait TimerFunction: Send + Sync {
    fn run(&mut self);
}
```

一种典型的实现方式是：新建一个零长的结构体，实现 TimerFunction 特性，然后在 run 函数中实现定时器要执行的操作。

9.6.4 4. 定时器 API

4.1. Timer 的 API

4.1.1. 创建一个定时器

```
pub fn new(timer_func: Box<dyn TimerFunction>, expire_jiffies: u64) -> Arc<Self>
```

参数

- timer_func：定时器需要执行的函数对应的结构体，其实现了 TimerFunction 特性
- expire_jiffies：定时器结束时刻（单位：**jiffies**）

返回

- 定时器结构体指针

4.1.2. 将定时器插入到定时器链表中

```
pub fn activate(&self)
```

4.2. 其余 API

若想要在.c 的模块中使用以下函数，请在函数名之前加上 rs_

4.2.1. 让进程休眠一段时间

```
pub fn schedule_timeout(mut timeout: i64) -> Result<i64, SystemError>
```

功能

让进程休眠 timeout 个 jiffies

参数

- timeout：需要休眠的时间（单位：**jiffies**）

返回值

- Ok(i64)：剩余需要休眠的时间（单位：**jiffies**）
- Err(SystemError)：错误码

4.2.2. 获取队列中第一个定时器的结束时间

```
pub fn timer_get_first_expire() -> Result<u64, SystemError>
```

功能

获取队列中第一个定时器的结束时间，即最早结束的定时器的结束时间

返回值

- Ok(i64)：最早结束的定时器的结束时间（单位：**jiffies**）
- Err(SystemError)：错误码

4.2.3. 获取当前系统时间

```
pub fn clock() -> u64
```

功能

获取当前系统时间（单位：**jiffies**）

4.2.4. 计算接下来 n 毫秒或者微秒对应的定时器时间片

4.2.4.1. 毫秒

```
pub fn next_n_ms_timer_jiffies(expire_ms: u64) -> u64
```

功能

计算接下来 n **毫秒**对应的定时器时间片

参数

- expire_ms：n 毫秒

返回值

对应的定时器时间片（单位：**毫秒**）

4.2.4.2. 微秒

```
pub fn next_n_us_timer_jiffies(expire_us: u64) -> u64
```

功能

计算接下来 n **微秒**对应的定时器时间片

参数

- expire_ms：n 微秒

返回值

对应的定时器时间片（单位：**微秒**）

9.6.5 5. 创建定时器实例

```
struct TimerExample {
    /// 结构体的成员对应函数的形参
    example_parameter: i32,
}
impl TimerExample {
    pub fn new(para: i32) -> Box<TimerExample> {
        return Box::new(TimerExample {
            example_parameter: para,
        });
    }
}
/// 为结构体实现TimerFunction特性
impl TimerFunction for TimerExample {
    /// TimerFunction特性中的函数run
    fn run(&mut self) {
        // 定时器需要执行的操作
        example_func(self.example_parameter);
    }
}
fn example_func(para: i32) {
    println!("para is {:?}", para);
}
fn main() {
    let timer_example: Box<TimerExample> = TimerExample::new(1);
    // 创建一个定时器
    let timer: Arc<Timer> = Timer::new(timer_example, 1);
    // 将定时器插入队列
    timer.activate();
}
```

CHAPTER 10

进程间通信

这里是 DragonOS 进程间通信（IPC）的说明文档。

10.1 Signal 信号

备注： 本文 Maintainer: 龙进

Email: longjin@RinGoTek.cn

信号是一种进程间通信机制，当我们发送一个信号给特定的进程，就能触发它的特定行为（例如退出程序，或者运行一些信号处理程序）。信号是发送到进程或同一进程内的特定线程的异步通知，用于通知它有事件发生。信号的常见用途是中断、挂起、终止或终止进程。发送信号时，操作系统会中断目标进程的正常执行流程以传递信号。可以在任何非原子指令期间中断执行。如果该进程之前注册了一个信号处理程序，则执行该例程。否则，将执行默认信号处理程序。

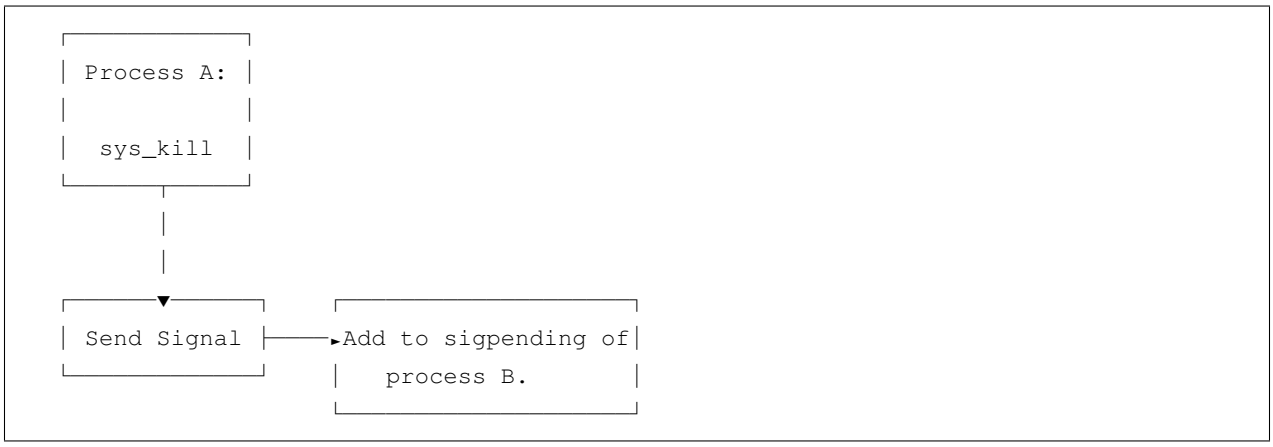
信号类似于中断，区别在于中断由 CPU 调解并由内核处理，而信号在内核中产生（也可通过系统调用让内核产生）并由各个进程或内核的默认处理函数处理。

10.1.1 1. 信号处理概要

1.1 信号发送

当进程 A 想发送信号给进程 B 的时候，使用 `kill(pid, signal)` 接口进行发送。然后陷入内核的 `sys_kill()` 函数中进行处理。然后内核将会把信号加入目标进程的 pcb 的 `sigpending` 中。

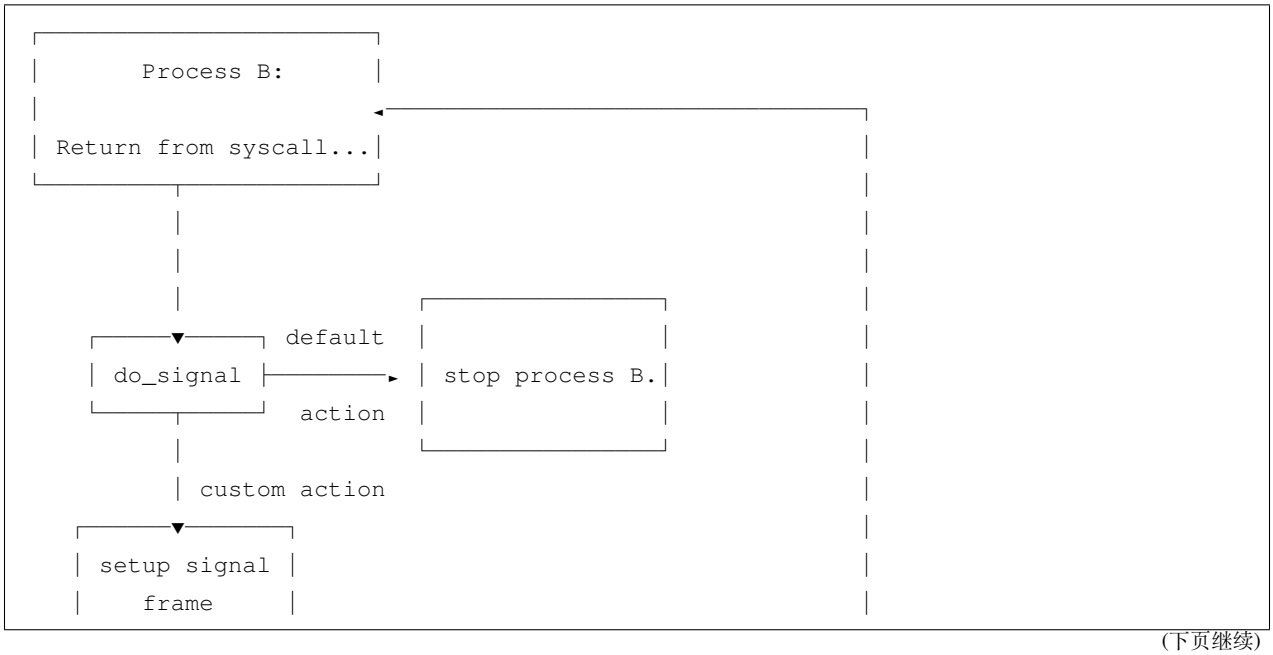
示意图如下：



1.2 信号处理

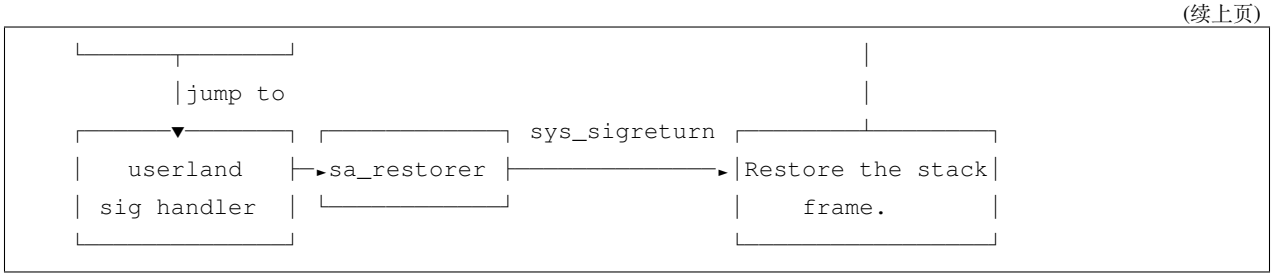
进程会在退出内核态的时候，跳转到 `do_signal()` 函数内，检查当前是否有需要被处理的信号，如果有的话，就会开启信号处理流程。

信号处理流程示意图：



(下页继续)

(续上页)



- 如果内核检查发现，进程没有指定信号处理函数，且信号处理动作不是“忽略”，那就会杀掉进程。
- 如果内核发现该信号没有被忽略，那么将会：
  - 备份当前内核栈
  - 设置信号处理的用户态栈帧
  - 回到用户态，执行信号处理函数
  - 信号处理函数结束之后，将会进入由 libc 提供的 __sa_restorer 中，发起 `sys_sigreturn()` 系统调用，回到内核态
  - 内核恢复处理信号之前的内核栈。
  - 信号处理流程结束，内核继续执行“返回用户态”的过程。
- 如果内核发现当前信号被忽略，那么就检查下一个信号。
- 发现没有任何需要处理的信号时，返回用户态。

10.1.2 2. 其他问题

暂无。

CHAPTER 11

内存管理

这里讲解了内存管理模块的一些设计及实现原理，以及相应的接口。

11.1 内存管理模块简介

11.1.1 1. 概述

DragonOS 实现了具有优秀架构设计的内存管理模块，对内核空间和用户空间的内存映射、分配、释放、管理等操作进行了封装，使得内核开发者可以更加方便地进行内存管理。

DragonOS 的内存管理模块主要由以下类型的组件组成：

- **硬件抽象层（MemoryManagementArch）** - 提供对具体处理器架构的抽象，使得内存管理模块可以在不同的处理器架构上运行
- **页面映射器（PageMapper）** - 提供对虚拟地址和物理地址的映射，以及页表的创建、填写、销毁、权限管理等操作。分为两种类型：内核页表映射器（KernelMapper）和用户页表映射器（位于具体的用户地址空间结构中）
- **页面刷新器（PageFlusher）** - 提供对页表的刷新操作（整表刷新、单页刷新、跨核心刷新）
- **页帧分配器（FrameAllocator）** - 提供对页帧的分配、释放、管理等操作。具体来说，包括 BumpAllocator、BuddyAllocator
- **小对象分配器** - 提供对小内存对象的分配、释放、管理等操作。指的是内核里面的 SlabAllocator （SlabAllocator 的实现目前还没有完成）

- **MMIO 空间管理器** - 提供对 MMIO 地址空间的分配、管理操作。(目前这个模块待进一步重构)
- **用户地址空间管理机制** - 提供对用户地址空间的管理。
  - VMA 机制 - 提供对用户地址空间的管理，包括 VMA 的创建、销毁、权限管理等操作
  - 用户映射管理 - 与 VMA 机制共同作用，管理用户地址空间的映射
- **系统调用层** - 提供对用户空间的内存管理系统调用，包括 mmap、munmap、mprotect、mremap 等
- **C 接口兼容层** - 提供对原有的 C 代码的接口，使得 C 代码能够正常运行。

11.2 内存分配指南

本文将讲述如何在内核中进行内存分配。在开始之前，请您先了解一个基本点：DragonOS 的内核使用 4KB 的页来管理内存，并且具有伙伴分配器和 slab 分配器。并且对用户空间、内核空间均具有特定的管理机制。

11.2.1 1. 安全的分配内存

在默认情况下，KernelAllocator 被绑定为全局内存分配器，它会根据请求分配的内存大小，自动选择使用 slab 还是伙伴分配器。因此，在内核中，使用 Rust 原生的内存分配函数，或者是创建一个 `Box` 对象等等，都是安全的。

11.2.2 2. 手动管理页帧

> **警告：请格外小心！**手动管理页帧脱离了 Rust 的内存安全机制，因此可能会造成内存泄漏或者是内存错误。

在某些情况下，我们需要手动分配页帧。例如，我们需要在内核中创建一个新的页表，或者是在内核中创建一个新的地址空间。这时候，我们需要手动分配页帧。使用 `LockedFrameAllocator` 的 `allocate()` 函数，能够分配在物理地址上连续的页帧。请注意，由于底层使用的是 buddy 分配器，因此页帧数目必须是 2 的 n 次幂，且最大大小不超过 1GB。

当需要释放页帧的时候，使用 `LockedFrameAllocator` 的 `deallocate()` 函数，或者是 `deallocate_page_frames()` 函数，能够释放在物理地址上连续的页帧。

当您需要映射页帧的时候，可使用 `KernelMapper::lock()` 函数，获得一个内核映射器对象，然后进行映射。由于 KernelMapper 是对 PageMapper 的封装，因此您在获取 KernelMapper 之后，可以使用 PageMapper 相关接口对内核空间的映射进行管理。

警告：千万不要使用 KernelMapper 去映射用户地址空间的内存，这会使得这部分内存脱离用户地址空间的管理，从而导致内存错误。

11.2.3 3. 为用户程序分配内存

在内核中，您可以使用用户地址空间结构体 (`AddressSpace`) 的 `mmap()`,`map_anonymous()` 等函数，为用户程序分配内存。这些函数会自动将用户程序的内存映射到用户地址空间中，并且会自动创建 VMA 结构体。您可以使用 `AddressSpace` 的 `munmap()` 函数，将用户程序的内存从用户地址空间中解除映射，并且销毁 VMA 结构体。调整权限等操作可以使用 `AddressSpace` 的 `mprotect()` 函数。

11.3 MMIO

MMIO 是“内存映射 IO”的缩写，它被广泛应用于与硬件设备的交互之中。

11.3.1 地址空间管理

DragonOS 中实现了 MMIO 地址空间的管理机制，本节将介绍它们。

为什么需要 MMIO 地址空间自动分配？

由于计算机上的很多设备都需要 MMIO 的地址空间，而每台计算机上所连接的各种设备的对 MMIO 地址空间的需求是不一样的。如果我们为每个类型的设备都手动指定一个 MMIO 地址，会使得虚拟地址空间被大大浪费，也会增加系统的复杂性。并且，我们在将来还需要为不同的虚拟内存区域做异常处理函数。因此，我们需要一套能够自动分配 MMIO 地址空间的机制。

这套机制提供了什么功能？

- 为驱动程序分配 4K 到 1GB 的 MMIO 虚拟地址空间
- 对于这些虚拟地址空间，添加到 VMA 中进行统一管理
- 可以批量释放这些地址空间

这套机制是如何实现的？

这套机制本质上是使用了伙伴系统来对 MMIO 虚拟地址空间进行维护。在 `mm/mm.h` 中指定了 MMIO 的虚拟地址空间范围，这个范围是 `0xffffa10000000000` 开始的 1TB 的空间。也就是说，这个伙伴系统为 MMIO 维护了这 1TB 的虚拟地址空间。

地址空间分配过程

1. 初始化 MMIO-mapping 模块，在 mmio 的伙伴系统中创建 512 个 1GB 的 `__mmio_buddy_addr_region`
2. 驱动程序使用 `mmio_create` 请求分配地址空间。
3. `mmio_create` 对申请的地址空间大小按照 2 的 n 次幂进行对齐，然后从 buddy 中申请内存地址空间
4. 创建 VMA，并将 VMA 标记为 `VM_IO|VM_DONTCOPY`。MMIO 的 vma 只绑定在 `initial_mm` 下，且不会被拷贝。
5. 分配完成

一旦 MMIO 地址空间分配完成，它就像普通的 vma 一样，可以使用 mmap 系列函数进行操作。

MMIO 的映射过程

在得到了虚拟地址空间之后，当我们尝试往这块地址空间内映射内存时，我们可以调用 `mm_map` 函数，对这块区域进行映射。

该函数会对 MMIO 的 VMA 的映射做出特殊处理。即：创建 `Page` 结构体以及对应的 `anon_vma`. 然后会将对应的物理地址，填写到页表之中。

MMIO 虚拟地址空间的释放

当设备被卸载时，驱动程序可以调用 `mmio_release` 函数对指定的 mmio 地址空间进行释放。

释放的过程中，`mmio_release` 将执行以下流程：

1. 取消 mmio 区域在页表中的映射。
2. 将释放 MMIO 区域的 VMA
3. 将地址空间归还给 mmio 的伙伴系统。

11.3.2 MMIO 的伙伴算法

伙伴的定义

同时满足以下三个条件的两个内存块被称为伙伴内存块：

1. 两个内存块的大小相同
2. 两个内存块的内存地址连续
3. 两个内存块由同一个大块内存分裂得到

伙伴算法

伙伴（buddy）算法的作用是维护以及组织大块连续内存块的分配和回收，以减少系统时运行产生的外部碎片。伙伴系统中的每个内存块的大小均为 2^n。在 DragonOS 中，伙伴系统内存池共维护了 1TB 的连续存储空间，最大的内存块大小为 $1G$，即 $2^{30}B$，最小的内存块大小为 $4K$，即 $2^{12}B$。

伙伴算法的核心思想是当应用申请内存时，每次都分配比申请的内存大小更大的最小内存块，同时分配出去的内存块大小为 2^nB。(e.g. 假设某应用申请了 $3B$ 内存，显然并没有整数值 n，使 $2^n = 3$，且 $3 \in [2^1, 2^2]$，所以系统会去取一块大小为 2^2B 的内存块，将它分配给申请的应用，本次申请内存操作顺利完成。)

那么当伙伴系统中没有如此“合适”的内存块时该怎么办呢？系统先会去寻找更大的内存块，如果找到了，则会将大内存块分裂成合适的内存块分配给应用。(e.g. 假设申请 $3B$ 内存，此时系统中比 $3B$ 大的最小内存块的大小为 $16B$，那么 $16B$ 会被分裂成两块 $8B$ 的内存块，一块放入内存池中，一块继续分裂成两块 $4B$ 的内存块。两块 $4B$ 的内存块，一块放入内存池中，一块分配给应用。至此，本次申请内存操作顺利完成。)

如果系统没有找到更大的内存块，系统将会尝试合并较小的内存块，直到符合申请空间的大小。(e.g. 假设申请 $3B$ 内存，系统检查内存池发现只有两个 $2B$ 的内存块，那么系统将会把这两个 $2B$ 的内存块合并成一块 $4B$ 的内存块，并分配给应用。至此，本次申请内存操作顺利完成。)

最后，当系统既没有找到大块内存，又无法成功合并小块内存时，就会通知应用内存不够，无法分配内存。

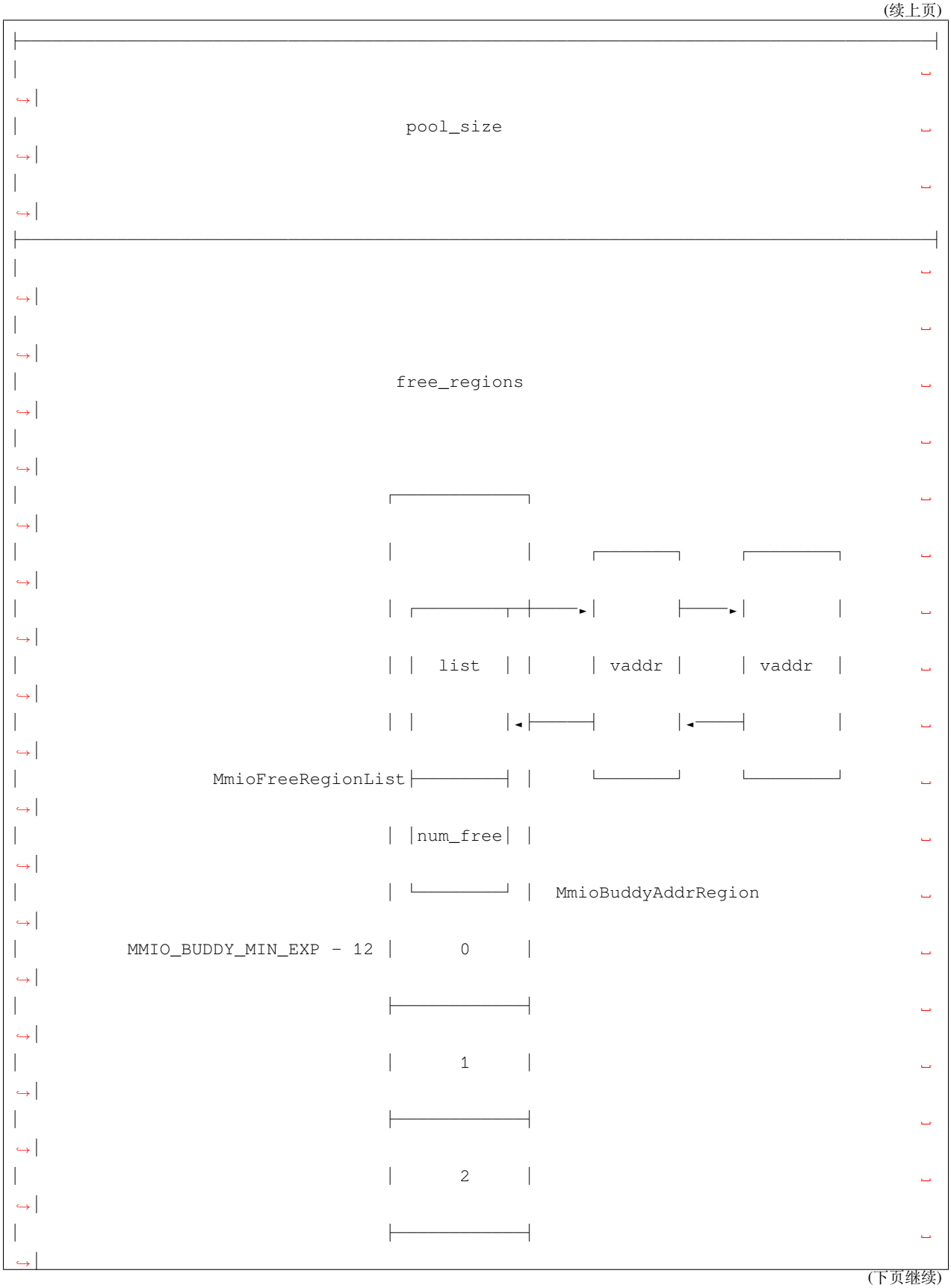
伙伴算法的数据结构

```
                                MmioBuddyMemPool

┌──────────────────────────────────────────────────────────────────────────────┐
│                                                                              │
↪|
│                              pool_start_addr                                 │
↪|
│                                                                              │
↪|
```

(下页继续)

(续上页)


pool_size
free_regions
MmioFreeRegionList
list
num_free
vaddr
vaddr
MmioBuddyAddrRegion
MMIO_BUDDY_MIN_EXP - 12
0
1
2

(下页继续)

(续上页)

```
│                                 │       3       │                              ␣
↪│
│                                 ├───────────────┤                              ␣
↪│
│                                 │      ...      │                              ␣
↪│
│                                 ├───────────────┤                              ␣
↪│
│                                 │      ...      │                              ␣
↪│
│                                 ├───────────────┤                              ␣
↪│
│          MMIO_BUDDY_MAX_EXP - 12 │      18       │                              ␣
↪│
│                                 └───────────────┘                              ␣
↪│
│                                                                                ␣
↪│
│                                                                                ␣
↪│
│                                                                                ␣
↪│
└────────────────────────────────────────────────────────────────────────────────┘
```

```rust
/// 最大的内存块为1G，其幂为30
const MMIO_BUDDY_MAX_EXP: u32 = PAGE_1G_SHIFT;
/// 最小的内存块为4K，其幂为12
const MMIO_BUDDY_MIN_EXP: u32 = PAGE_4K_SHIFT;
/// 内存池数组的大小为18
const MMIO_BUDDY_REGION_COUNT: u32 = MMIO_BUDDY_MAX_EXP - MMIO_BUDDY_MIN_EXP + 1;

/// buddy内存池
pub struct MmioBuddyMemPool {
    /// 内存池的起始地址
    pool_start_addr: u64,
    /// 内存池大小：初始化为1TB
    pool_size: u64,
    /// 空闲内存块链表数组
    /// MMIO_BUDDY_REGION_COUNT = MMIO_BUDDY_MAX_EXP - MMIO_BUDDY_MIN_EXP + 1
    free_regions: [SpinLock<MmioFreeRegionList>; MMIO_BUDDY_REGION_COUNT as usize],
}

/// 空闲内存块链表结构体
```

(下页继续)

(续上页)

```
pub struct MmioFreeRegionList {
    /// 存储了空闲内存块信息的结构体的链表
    list: LinkedList<Box<MmioBuddyAddrRegion>>,
    /// 当前链表空闲块的数量
    num_free: i64,
}

/// mmio伙伴系统内部的地址区域结构体
pub struct MmioBuddyAddrRegion {
    /// 内存块的起始地址
    vaddr: u64,
}
```

设计思路

DragonOS 中，使用 `MmioBuddyMemPool` 结构体作为 buddy（为表述方便，以下将伙伴算法简称为 buddy）内存池的数据结构，其记录了内存池的起始地址（pool_start_addr）以及内存池中内存块的总大小（pool_size），同时其维护了大小为 `MMIO_BUDDY_REGION_COUNT` 的双向链表数组（free_regions）。`free_regions` 中的各个链表维护了若干空闲内存块（MmioBuddyAddrRegion）。

`free_regions` 的下标（index）与内存块的大小有关。由于每个内存块大小都为 2^n bytes，那么可以令 $exp = n$。index 与 exp 的换算公式如下：$index = exp - 12$。e.g. 一个大小为 2^{12} bytes 的内存块，其 $exp = 12$，使用上述公式计算得 $index = 12 - 12 = 0$，所以该内存块会被存入 `free_regions[0].list` 中。通过上述换算公式，每次取出或释放 2^n 大小的内存块，只需要操作 `free_regions[n -12]` 即可。DragonOS 中，buddy 内存池最大的内存块大小为 $1G = 2^{30} bytes$，最小的内存块大小为 $4K = 2^{12} bytes$，所以 $index \in [0, 18]$。

作为内存分配机制，buddy 服务于所有进程，为了解决在各个进程之间实现 free_regions 中的链表数据同步的问题，`free_regions` 中的链表类型采用加了自旋锁（SpinLock）的空闲内存块链表（MmioFreeRegionList），`MmioFreeRegionList` 中封装有真正的存储了空闲内存块信息的结构体的链表（list）和对应链表长度（num_free）。有了自选锁后，同一时刻只允许一个进程修改某个链表，如取出链表元素（申请内存）或者向链表中插入元素（释放内存）。

`MmioFreeRegionList` 中的元素类型为 `MmioBuddyAddrRegion` 结构体，`MmioBuddyAddrRegion` 记录了内存块的起始地址（vaddr）。

伙伴算法内部 api

P.S 以下函数均为 MmioBuddyMemPool 的成员函数。系统中已经创建了一个 MmioBuddyMemPool 类型的全局引用 `MMIO_POOL`，如要使用以下函数，请以 `MMIO_POOL.xxx()` 形式使用，以此形式使用则不需要传入 self。

函数名	描述
__create_region(&self, vaddr)	将虚拟地址传入，创建新的内存块地址结构体
__give_back_block(&self, vaddr, exp)	将地址为 vaddr，幂为 exp 的内存块归还给 buddy
__buddy_split(&self,region,exp,list_guard)	将给定大小为 2^{exp} 的内存块一分为二，并插入内存块大小为 2^{exp-1} 的链表中
__query_addr_region(&self,exp,list_guard)	从 buddy 中申请一块大小为 2^{exp} 的内存块
mmio_buddy_query_addr_region(&self,exp)	对 query_addr_region 进行封装，**请使用这个函数，而不是 __query_addr_region**
__buddy_add_region_obj(&self,region,list_guard)	往指定的地址空间链表中添加一个内存块
__buddy_block_vaddr(&self, vaddr, exp)	根据地址和内存块大小，计算伙伴块虚拟内存的地址
__pop_buddy_block(&self, vaddr,exp,list_guard)	寻找并弹出指定内存块的伙伴块
__buddy_pop_region(&self, list_guard)	从指定空闲链表中取出内存区域
__buddy_merge(&self,exp,list_guard,high_list_guard)	合并所有 2^{exp} 大小的内存块
__buddy_merge_blocks(&self,region_1,region_2,exp,high_list_guard)	合并两个**已经从链表中取出的**内存块

伙伴算法对外 api

函数名	描述
__mmio_buddy_init()	初始化 buddy 系统，**在 mmio_init() 中调用，请勿随意调用**
__exp2index(exp)	将 2^{exp} 的 exp 转换成内存池中的数组的下标（index）
mmio_create(size,vm_flags,res_vaddr,res_length)	创建一块根据 size 对齐后的大小的 mmio 区域，并将其 vma 绑定到 initial_mm
mmio_release(vaddr, length)	取消地址为 vaddr，大小为 length 的 mmio 的映射并将其归还到 buddy 中

CHAPTER 12

文件系统

DragonOS 的文件系统模块由 VFS（虚拟文件系统）及具体的文件系统组成。

todo: 由于文件系统模块重构，文档暂时不可用，预计在 2023 年 4 月 10 日前补齐。

备注： 本文作者: 龙进

Email: longjin@DragonOS.org

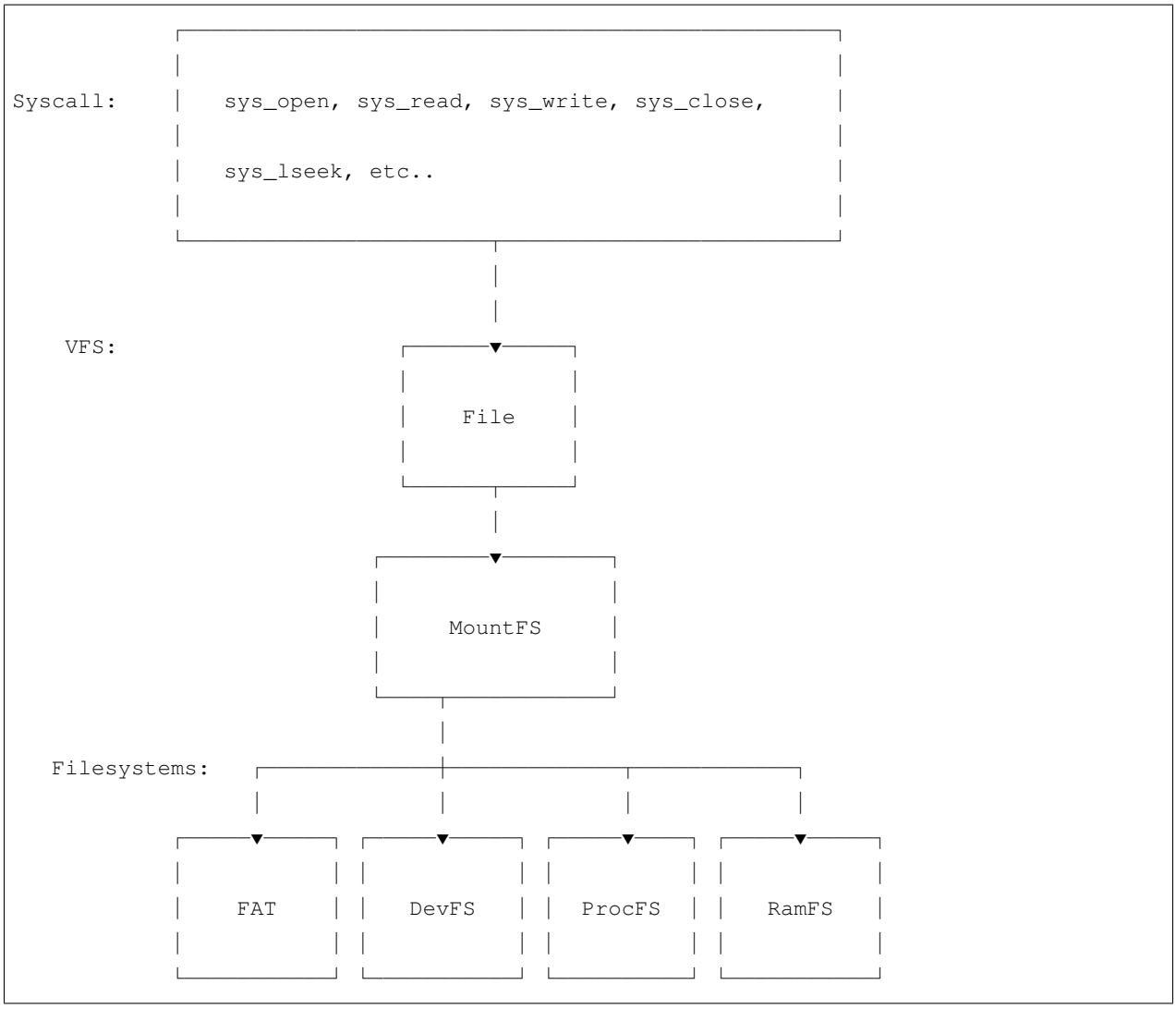
12.1 概述

在本文中，我们将介绍 DragonOS 文件系统的架构设计。

12.1.1 总览

如下图所示，DragonOS 的文件系统相关的机制主要包括以下几个部分：

- 系统调用接口
- 虚拟文件系统
  - 文件抽象（File）
  - 挂载文件系统（MountFS）
- 具体的文件系统



12.1.2 系统调用接口

DragonOS 的文件系统相关的系统调用接口主要包括以下几个：

- `sys_open`：打开文件
- `sys_read`：读取文件
- `sys_write`：写入文件
- `sys_close`：关闭文件
- `sys_lseek`：定位文件指针
- `sys_mkdir`：创建目录
- `sys_unlink_at`：删除文件或目录（通过参数 `flag` 区分到底是删除文件还是目录）
- `sys_ioctl`：控制设备（未实现）

- `sys_fstat`：获取文件状态（未实现）
- `sys_fsync`：同步文件（未实现）
- `sys_ftruncate`：截断文件（未实现）
- `sys_fchmod`：修改文件权限（未实现）
- 其他系统调用接口（未实现）

  关于接口的具体含义，可以参考*DragonOS 系统调用接口*。

12.1.3 虚拟文件系统（VFS）

VFS 是 DragonOS 文件系统的核心，它提供了一套统一的文件系统接口，使得 DragonOS 可以支持多种不同的文件系统。VFS 的主要功能包括：

- 提供统一的文件系统接口
- 提供文件系统的挂载和卸载机制（MountFS）
- 提供文件抽象（File）
- 提供文件系统的抽象（FileSystem）
- 提供 IndexNode 抽象
- 提供文件系统的缓存、同步机制（尚未实现）

  关于 VFS 的详细介绍，请见*DragonOS 虚拟文件系统*。

12.1.4 具体的文件系统

DragonOS 目前支持的文件系统包括：

- FAT 文件系统（FAT12、FAT16、FAT32）
- DevFS
- ProcFS
- RamFS

12.2 VFS 虚拟文件系统

在 DragonOS 中，VFS 作为适配器，遮住了具体文件系统之间的差异，对外提供统一的文件操作接口抽象。

VFS 是 DragonOS 文件系统的核心，它提供了一套统一的文件系统接口，使得 DragonOS 可以支持多种不同的文件系统。VFS 的主要功能包括：

- 提供统一的文件系统接口

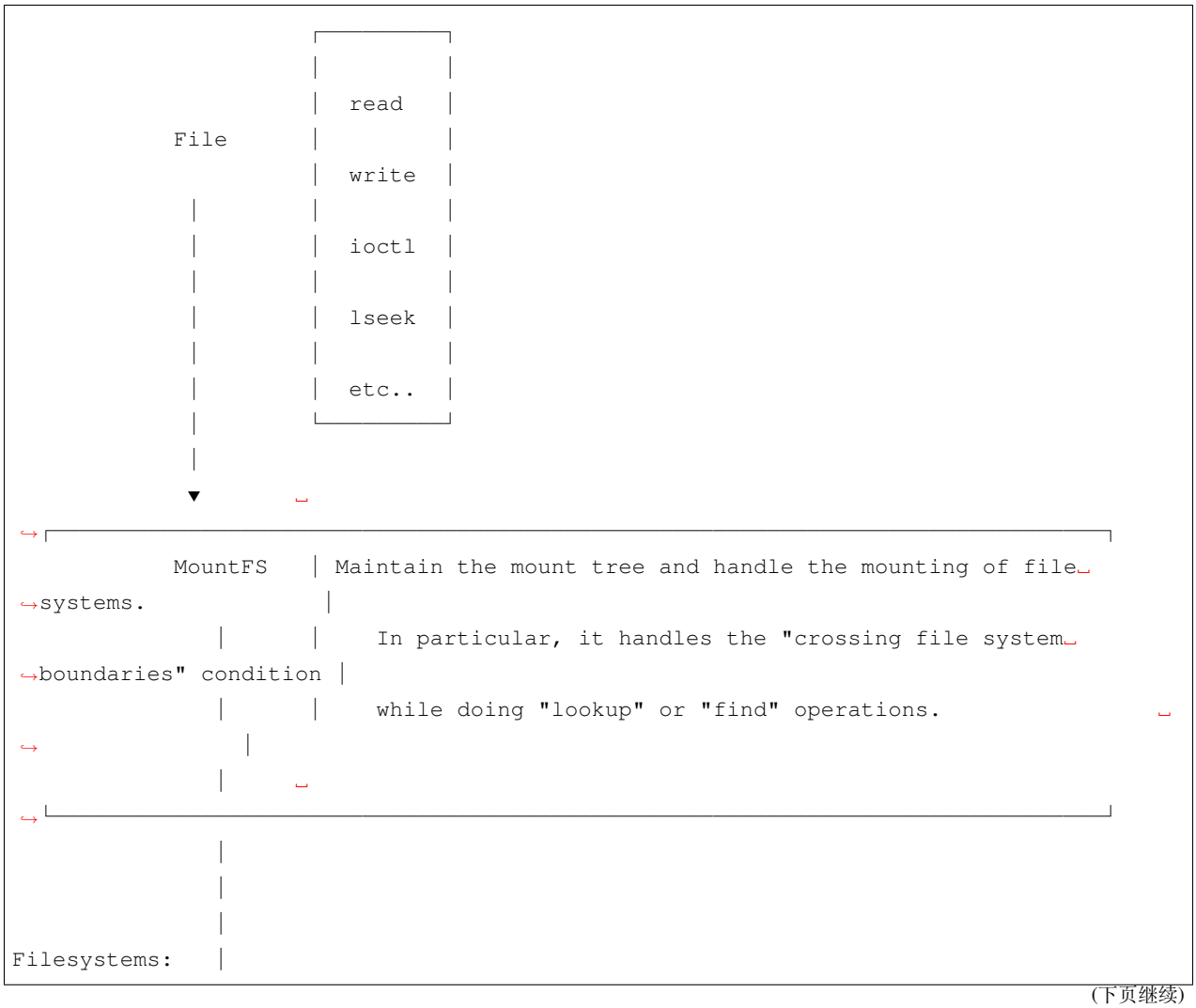
- 提供文件系统的挂载和卸载机制（MountFS）
- 提供文件抽象（File）
- 提供文件系统的抽象（FileSystem）
- 提供 IndexNode 抽象
- 提供文件系统的缓存、同步机制（尚未实现）

备注： 本文作者: 龙进

Email: longjin@DragonOS.org

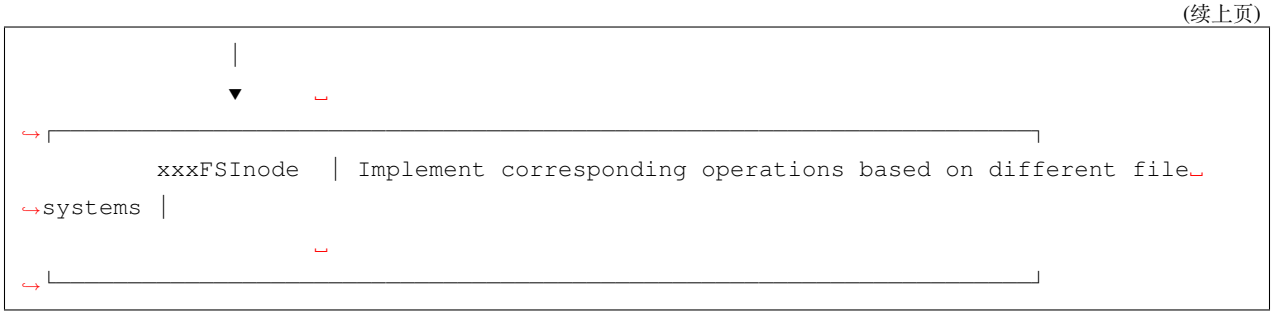
12.2.1 设计

VFS 的架构设计如下图所示：



(下页继续)

(续上页)



1. File

File 结构体是 VFS 中最基本的抽象，它代表了一个打开的文件。每当进程打开了一个文件，就会创建一个 File 结构体，用于维护该文件的状态信息。

2. Traits

对于每个具体文件系统，都需要实现以下的 trait：

- FileSystem：表明某个 struct 是一个文件系统
- IndexNode：表明某个 struct 是一个索引节点

一般情况下，FileSystem 和 IndexNode 是一对一的关系，也就是，一个文件系统对应一种 IndexNode。但是，对于某些特殊的文件系统，比如 DevFS，根据不同的设备类型，会有不同的 IndexNode，因此，FileSystem 和 IndexNode 是一对多的关系。

3. MountFS

挂载文件系统虽然实现了 FileSystem 和 IndexNode 这两个 trait，但它并不是一个“文件系统”，而是一种机制，用于将不同的文件系统挂载到同一个文件系统树上. 所有的文件系统要挂载到文件系统树上，都需要通过 MountFS 来完成。也就是说，挂载树上的每个文件系统结构体的外面，都套了一层 MountFS 结构体。

对于大部分的操作，MountFS 都是直接转发给具体的文件系统，而不做任何处理。同时，为了支持跨文件系统的操作，比如在目录树上查找，每次 lookup 操作或者是 find 操作，都会通过 MountFSInode 的对应方法，判断当前 inode 是否为挂载点，并对挂载点进行特殊处理。如果发现操作跨越了具体文件系统的边界，MountFS 就会将操作转发给下一个文件系统，并执行 Inode 替换。这个功能的实现，也是通过在普通的 Inode 结构体外面，套一层 MountFSInode 结构体来实现的。

12.2.2 VFS API 文档

12.3 SysFS

备注： 本文作者：黄厅

Email: huangting@DragonOS.org

12.3.1 1. SysFS 和设备驱动模型

1.1. 设备、驱动、总线等彼此之间关系错综复杂

如果想让内核运行流畅，那就必须为每个模块编码实现这些功能。如此一来，内核将变得非常臃肿、冗余。而设备模型的理念即是将这些代码抽象成各模块共用的框架，这样不但代码简洁了，也可让设备驱动开发者摆脱这本让人头痛但又必不可少的一劫，将有限的精力放于设备差异性的实现。

设备模型恰是提供了一个模板，一个被证明过的最优的思路和流程，这减少了开发者设计过程中不必要的错误，也给以后的维护扫除了障碍。

1.2. sysfs 是一个基于内存的文件系统，它的作用是将内核信息以文件的方式提供给用户程序使用。

sysfs 可以看成与 proc,devfs 和 devpty 同类别的文件系统，该文件系统是虚拟的文件系统，可以更方便对系统设备进行管理。它可以产生一个包含所有系统硬件层次视图，与提供进程和状态信息的 proc 文件系统十分类似。sysfs 把连接在系统上的设备和总线组织成为一个分级的文件，它们可以由用户空间存取，向用户空间导出内核的数据结构以及它们的属性。

12.3.2 2. DragosOS 中的设备驱动模型

2.1 由设备和驱动构成基本元素

2.1.1. 设备

```
/// @brief: 所有设备都应该实现该trait
pub trait Device: Any + Send + Sync + Debug {}
```

DragonOS 采用全局设备管理器管理系统中所有的设备。

```
/// @brief Device管理器
#[derive(Debug, Clone)]
```

(下页继续)

(续上页)

```
pub struct DeviceManager {
    devices: BTreeMap<IdTable, Arc<dyn Device>>, // 所有设备
    sys_info: Option<Arc<dyn IndexNode>>,  // sys information
}
```

2.1.2. 驱动

```
/// @brief: 所有驱动驱动都应该实现该trait
pub trait Driver: Any + Send + Sync + Debug {}
```

同样的，驱动也使用全局的驱动管理器来管理

```
/// @brief: 驱动管理器
#[derive(Debug, Clone)]
pub struct DriverManager {
    drivers: BTreeMap<IdTable, Arc<dyn Driver>>, // 所有驱动
    sys_info: Option<Arc<dyn IndexNode>>, // sys information
}
```

2.2. 总线

总线属于设备的一种类型，同样需要驱动来初始化，同时由于总线的特殊性，使用全局的总线管理器来进行管理。

```
/// @brief: 总线驱动trait，所有总线驱动都应实现该trait
pub trait BusDriver: Driver {}

/// @brief: 总线设备trait，所有总线都应实现该trait
pub trait Bus: Device {}

/// @brief: 总线管理结构体
#[derive(Debug, Clone)]
pub struct BusManager {
    buses: BTreeMap<IdTable, Arc<dyn Bus>>,          // 总线设备表
    bus_drvs: BTreeMap<IdTable, Arc<dyn BusDriver>>, // 总线驱动表
    sys_info: Option<Arc<dyn IndexNode>>,            // 总线inode
}
```

可以看到，每个管理器中均存在 sys_info，设备模型通过该成员与 sysfs 建立联系，sys_info 指向 sysfs 中唯一的 inode。对于 device 而言，对应 sysfs 下的 devices 文件夹，其他亦是如此。

12.3.3 3. 驱动开发如何进行

以平台总线 platform 为例，platform 总线是一种虚拟总线，可以对挂载在其上的设备和驱动进行匹配，并驱动设备。该总线是一类设备，同时也是一类总线，编程时需要创建该设备实例，并为设备实例实现 Device trait 和 Bus trait，以表明该结构是一类总线设备。同时，应该实现总线上的匹配规则，不同的总线匹配规则不同，该总线采用匹配表方式进行匹配，设备和驱动都应该存在一份匹配表，表示驱动支持的设备以及设备支持的驱动。

```
pub struct CompatibleTable(BTreeSet<&'static str>);
```

对于 bus 设备而言，需要调用 bus_register，将 bus 注册进系统，并在 sysfs 中可视化。

```
/// @brief: 总线注册，将总线加入全局总线管理器中，并根据id table在sys/bus和sys/
↪devices下生成文件夹
/// @parameter bus: Bus设备实体
/// @return: 成功:()   失败:DeviceError
pub fn bus_register<T: Bus>(bus: Arc<T>) -> Result<(), DeviceError> {
    BUS_MANAGER.add_bus(bus.get_id_table(), bus.clone());
    match sys_bus_register(&bus.get_id_table().to_name()) {
        Ok(inode) => {
            let _ = sys_bus_init(&inode);
            return device_register(bus);
        }
        Err(_) => Err(DeviceError::RegisterError),
    }
}
```

通过 bus_register 源码可知，该函数不仅在 sysfs/bus 下生成总线文件夹，同时内部调用 device_register，该函数将总线加入设备管理器中，同时在 sys/devices 下生成设备文件夹。

12.4 KernFS

备注： Maintainer:

- 龙进 longjin@dragonos.org

12.4.1 1. 简介

KernFS 是一个伪文件系统，它充当其它内核文件系统的容器，面向用户提供文件接口。其核心功能就是，当 kernfs 的文件被读/写或者触发回调点的时候，将会对预设的回调函数进行调用，触发其它内核文件系统的操作。

这种设计使得 SysFS 和文件系统的基本操作解耦，KernFS 作为 SysFS 的承载物，使得 SysFS 能更专注于 KObject 的管理，让代码更加优雅。

在未来，DragonOS 的内核子系统，或者其它的内核文件系统，可以使用 KernFS 作为文件系统操作的承载物，让系统管理的逻辑与具体的文件系统操作解除耦合。

12.4.2 2. 使用方法

以 SysFS 为例，新创建一个 KernFS 实例，作为 SysFS 的文件系统接口，然后挂载到`/sys`目录下。接着 sysfs 实现上层逻辑，管理 KObject，每个上层的 Kobject 里面都需要包含 KernFSInode。并且通过设置 KernFSInode 的 PrivateData，使得 KernFS 能够根据 Inode 获取到其指向的 KObject 或者 sysfs 的 attribute。并且在创建 KernFSInode 的时候，为具体的 Inode 传入不同的 callback，以此实现“不同的 Inode 在读写时能够触发不同的回调行为”。

当发生回调时，KernFS 会把回调信息、私有信息传入到回调函数中，让回调函数能够根据传入的信息，获取到对应的 KObject 或者 sysfs 的 attribute，从而实现 sysfs 提供的高层功能。

从上述描述我们能够看出：KernFS 就是通过存储上层文件系统的回调函数、回调信息，来实现“把具体文件操作与高层管理逻辑进行解耦”的目的。

CHAPTER 13

内核调试模块

这里是 DragonOS 的内核调试模块文档。

13.1 内核栈 traceback

13.1.1 简介

内核栈 traceback 的功能位于 `kernel/debug/traceback/`文件夹中。为内核态提供 traceback 的功能，打印调用栈到屏幕上。

13.1.2 API

`void traceback(struct pt_regs * regs)`

作用

该接口定义于 `kernel/debug/traceback/traceback.h` 中，将会对给定内核栈进行 traceback，并打印跟踪结果到屏幕上。

参数

regs

要开始追踪的第一层内核栈栈帧（也就是栈的底端）

13.1.3 实现原理

当内核第一次链接之后，将会通过 Makefile 中的命令，运行 `kernel/debug/kallsyms` 程序，提取内核文件的符号表，然后生成 `kernel/debug/kallsyms.S`。该文件的 rodata 段中存储了 text 段的函数的符号表。接着，该文件将被编译为 `kallsyms.o`。最后，Makefile 中再次调用 `ld` 命令进行链接，将 kallsyms.o 链接至内核文件。

当调用 `traceback` 函数时，其将遍历该符号表，找到对应的符号并输出。

13.1.4 未来发展方向

- 增加写入到日志文件的功能

13.2 如何使用 GDB 调试内核

13.2.1 前言

GDB 是一个功能强大的开源调试工具，能够帮助您更好的诊断和修复程序中的错误。 它提供了一套丰富的功能，使您能够检查程序的执行状态、跟踪代码的执行流程、查看和修改变量的值、分析内存状态等。它可以与编译器配合使用，以便您在调试过程中访问程序的调试信息。

此教程将告诉您如何在 DragonOS 中使用 `rust-gdb` 来调试内核，包括如何开始调试以及相应的调试命令。

备注： 如果您已经熟悉了 `rust-gdb` 的各种命令，那您只需要阅读此教程的第一部分即可。

13.2.2 1. 从何开始

1.1 准备工作

在您开始调试内核之前，需要在/Kernel/Cargo.toml 中开启调试模式，将 Cargo.toml 中的 `debug = false` 更改为 `debug = true`。

```
debug = false
```

更改为

```
debug = true
```

1.2 运行 DragonOS

准备工作完成后，您就可以编译、运行 DragonOS 来开展后续的调试工作了。 在 DragonOS 根目录中开启终端，使用 `make run` 即可开始编译运行 DragonOS, 如需更多编译命令方面的帮助，详见

构建 DragonOS。

1.3 运行 GDB

当 DragonOS 开始运行后，您就可以启动 GDB 开始调试了。

您只需要开启一个新的终端，运行 `make gdb` 即可运行 GDB 调试器。

```
 make gdb
rust-gdb -n -x tools/.gdbinit
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

--Type <RET> for more, q to quit, c to continue without paging--
```

备注： 若出现以上信息，输入 c 再回车即可。

13.2.3 2. 调试

2.1 开始

当以上步骤完成后，就已经可以开始调试了。

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0xffff8000001f8f63 in ?? ()
(gdb)
```

备注： GDB 输出的信息中 `0xffff8000001f8f63 in ?? ()` 表明 DragonOS 还在引导加载的过程中。

输入 `continue` 或者 `c`，程序将继续执行。

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0xffff8000001f8f63 in ?? ()
(gdb) continue
Continuing.
```

在 DragonOS 运行时，您可以随时按下 `Ctrl+C` 来发送中断信息。来查看内核当前状态。

```
(gdb) continue
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
0xffff800000140c21 in io_in8 (port=113) at common/glib.h:136
136         __asm__ __volatile__("inb   %%dx,   %0      \n\t"
(gdb)
```

2.2 设置断点和监视点

设置断点和监视点是程序调试中最基础的一步。

- **设置断点**

  您可以使用 `break` 或者 `b` 命令来设置断点。

  关于 `break` 或者 `b` 命令的使用:

```
b <line_number> #在当前活动源文件的相应行号打断点

b <file>:<line_number> #在对应文件的相应行号打断点

b <function_name> #为一个命名函数打断点
```

- **设置监视点**

  您可以使用 `watch` 命令来设置监视点

```
watch <variable> # 设置对特定变量的监视点,将在特定变量发生变化的时候触发断点

watch <expression> # 设置对特定表达式的监视点，比如watch␣
↪*(int*)0x12345678会在内存地址0x12345678处
                   # 的整数值发生更改时触发断点。
```

- **管理断点与监视点**

  当我们打上断点之后，我们该如何查看我们所有的断点信息呢?

  您可以通过 `info b`，`info break` 或者 `info breakpoints` 来查看所有的断点信息:

```
(gdb) b 309
Breakpoint 12 at 0xffff8000001f8f16: file /home/heyicong/.cargo/registry/src/mirrors.
↪tuna.tsinghua.edu.cn-df7c3c540f42cdbd/thingbuf-0.1.4/src/lib.rs, line 315.
(gdb) watch slots
Watchpoint 13: slots
(gdb) info b
Num     Type           Disp Enb Address            What
12      breakpoint     keep y   0xffff8000001f8f16 in thingbuf::Core::pop_ref<u8>
                                                   at /home/heyicong/.cargo/registry/
↪src/mirrors.tuna.tsinghua.edu.cn-df7c3c540f42cdbd/thingbuf-0.1.4/src/lib.rs:315
13      watchpoint     keep y                      slots
(gdb)
```

以上信息中，编号为 12 的断点即是我们在活动源文件 309 行打的断点，若其 `Address` 为`<MULTIPLE>`，则表示在多个地址上存在相同的断点位置。这在循环中是非常常见的情况。编号为 13 的便是我们对 `slots` 变量设置的监视点。

我们可以通过以下命令对断点或者监视点进行操作：

```
delete <breakpoint#> # 或 d <breakpoint#>␣
↪删除对应编号的断点，在您不再需要使用这个断点的时候可以通过此命令删除断点
delete <watchpoint#> # 或 d <watchpoint##>␣
↪删除对应编号的监视点，在您不再需要使用这个监视点的时候可以通过此命令删除监视点

disable <breakpoint#> #␣
↪禁用对应编号的断点，这适合于您只是暂时不需要使用这个断点时使用，当您禁用一个断点，下
                      # 次程序运行到该断点处将不会停下来
disable <watchpoint#> #␣
↪禁用对应编号的监视点，这适合于您只是暂时不需要使用这个监视点时使用

enable <breakpoint#> # 启用对应编号的断点
enable <watchpoint#> # 启用对应编号的监视点

#clear命令
clear # 清除当前活动源文件的断点以及监视点
clear <point_number> # 清除对应编号的所有断点或监视点，这与delete行为是一致的
clear <file> # 清除指定文件的所有断点与监视点
```

13.2.4 2.3 变量和内存查看

- **print 和 display**

  您可以通过 `print` 或者 `p` 来打印变量值。

  `print` 命令用于打印变量或表达式的值。它允许您在调试过程中查看程序中的数据。

```
print <variable> # 打印对应变量名的值，例如：print my_variable 或者 p my_variable

print <expression> # 打印合法表达式的值，例如：print a+b 或者 p a+b

# 示例输出
(gdb) print order
$3 = core::sync::atomic::Ordering::SeqCst
```

备注： 如果您不仅想打印值，还想显示更多详细信息（例如类型信息），可以使用 ptype 命令。

您可以使用 `display` 命令来持续追踪变量或者表达式,`display` 命令用于设置需要持续跟踪并在每次程序停止时显示的表达式。它类似于 print 命令，但与 print 不同的是，display 命令在每次程序停止时自动打印指定表达式的值，而无需手动输入命令。

```
display <variable> # 打印对应变量名的值，例如：display my_variable

display <expression> # 打印合法表达式的值，例如：display a+b

# 示例输出
(gdb) display order
1: order = core::sync::atomic::Ordering::SeqCst #其中1表示display编号，
                                                #您可以通过info␣
↪display命令来查看所有display编号
```

备注： 一旦您设置了 display 命令，每当程序停止（例如，在断点处停止）时，GDB 将自动打印指定表达式的值。

display 命令非常有用，因为它允许您在调试过程中持续监视表达式的值，而无需每次都手动输入 print 命令。它特别适用于那些您希望持续跟踪的变量或表达式。

要取消已设置的 display 命令并停止自动显示表达式的值，可以使用 undisplay 命令：

```
undisplay <display编号> # 如果不指定<display编号>，则将取消所有已设置的display命令，
                        # 您可以通过info display命令来查看所有display编号
```

备注： 请注意，print 和 display 命令只会在程序暂停执行时评估变量或表达式的值。如果程序正在运行，您需要通过设置断点或使用其他调试命令来暂停程序，然后才能使用 print 命令查看数据的值,display 命令设置的值将会在程序暂停时自动输出。

- **输出格式**

  您可以设置输出格式来获取更多您需要的信息, 例如：`print /a var`

  参考至GDB Cheat Sheet

```
Format
a Pointer.
c Read as integer, print as character.
d Integer, signed decimal.
f Floating point number.
o Integer, print as octal.
s Try to treat as C string.
t Integer, print as binary (t = „two “).
u Integer, unsigned decimal.
x Integer, print as hexadecimal.
```

2.4 查看调用堆栈

- **查看调用栈**

当程序在断点处暂停时，应该怎样追踪程序行为呢?

您可以通过 `backtarce` 命令来查看调用栈。backtrace 命令用于打印当前调用栈的回溯信息。它显示了程序在执行过程中所有活动的函数调用链，包括每个函数的名称、参数和源文件中的行号。

```
# 示例输出
(gdb) backtrace
#0  function1 (arg1=10, arg2=20) at file1.c:15
#1  function2 () at file2.c:25
#2  xx () at xx.c:8
```

每一行回溯信息都以 #<frame_number> 开头，指示帧的编号。然后是函数名和参数列表，最后是源文件名和行号。通过查看回溯信息，您可以了解程序在哪些函数中执行，以及每个函数在调用栈中的位置。这对于调试程序和定位问题非常有用。

- **切换堆栈**

您可以通过 `frame` 或者 `f` 命令来切换对应的栈帧获取更多信息以及操作。

```
frame <frame_number>
f <frame_number>
```

除了简单地执行 backtrace 命令，还可以使用一些选项来自定义回溯信息的输出。例如:

```
backtrace full                          #显示完整的符号信息，包括函数参数和局部变量。
backtrace <frame_count>                 #限制回溯信息的帧数，只显示指定数量的帧。
backtrace <frame_start>-<frame_end>     #指定要显示的帧范围。
backtrace thread <thread_id>            #显示指定线程的回溯信息。
```

2.5 多核心

在调试内核时，您可能需要查看各个核心的运行状态。

您可以通过 `info threads` 命令来查看各个核心的运行状态

```
(gdb) info threads
  Id   Target Id                    Frame
  1    Thread 1.1 (CPU#0 [halted ]) 0xffff800000140a3e in Start_Kernel () at main.
↪c:227
* 2    Thread 1.2 (CPU#1 [running]) thingbuf::Core::pop_ref<u8> ()
    at /home/heyicong/.cargo/registry/src/mirrors.tuna.tsinghua.edu.cn-
↪df7c3c540f42cdbd/thingbuf-0.1.4/src/lib.rs:315
(gdb)
```

您可以使用 thread <thread_id> 命令切换到指定的核心上下文，以便查看和调试特定核心的状态。例如：

```
(gdb) thread 1
[Switching to thread 1 (Thread 1.1)]
#0  0xffff800000140a3e in Start_Kernel () at main.c:227
227                 hlt();
```

2.6 更多

接下来，我将为您介绍更多您可能在调试中能够使用的命令：

```
step                    #或者s,
↪逐行执行程序，并进入到函数调用中。可以在step命令后加执行次数，例：step 3␣
↪表示要连续执行3个步骤
step <function>         #进入指定的函数，并停止在函数内的第一行。

next                    #或者n,
↪逐行执行程序，但跳过函数调用，直接执行函数调用后的下一行代码。

↪#它允许你在不进入函数内部的情况下执行代码，从而快速跳过函数调用的细节。
                        #同样，next也可以在命令后加执行次数

finish                  #用于从当前函数中一直执行到函数返回为止，并停在调用该函数的地方。

↪#它允许你快速执行完当前函数的剩余部分，并返回到调用函数的上下文中。

continue                #用于继续程序的执行，直到遇到下一个断点或
                        #程序正常结束或者程序暂停。

quit                    #退出调试

list
↪#或者l，显示当前活动源文件源代码的片段，以及当前执行的位置。
list <filename>:<function>      #显示<filename>文件里面的<funtion>函数的源代码片段
list <filename>:<line_number>   #显示<filename>文件里面的<line_number>附近的源代码片段
list <first>,<last>             #显示当前活动源文件的<first>至<last>之间的源代码片段
set listsize <count>
↪#设置list命令显示的源代码行数。默认情况下，list命令显示当前行和其周围的几行代码。

info args                       #显示当前函数的参数及其值
info breakpoints                #显示断点以及监视点信息
info display                    #显示当前设置的display列表
info locals                     #显示当前函数/栈帧中的局部变量及其值
```

(下页继续)

(续上页)

```
info sharedlibrary                #显示当前已加载的共享库（shared library）信息
info signals
↪#显示当前程序所支持的信号信息。它可以列出程序可以接收和处理的不同信号的列表。
info threads                      #显示各个核心/线程信息，它可以列出当前正在运行的核心/
↪线程以及它们的状态。

show directories
↪#显示当前源代码文件的搜索路径列表。这些搜索路径决定了GDB在查找源代码文件时的搜索范围。
show listsize
↪#显示打印源代码时的上下文行数。它确定了在使用list命令（或其简写形式l）时显示的源代码行数。

whatis variable_name
↪#查看给定变量或表达式的类型信息。它可以帮助你了解变量的数据类型。
ptype
↪#显示给定类型或变量的详细类型信息。它可以帮助你了解类型的结构和成员。
                                  #相较于whatis命令，ptype命令更加详细。

set var <variable_name>=<value> #设置变量值

return <expression>               #强制使当前函数返回设定值                              ␣
↪
```

13.2.5 最后

现在，您已经可以使用 rust-gdb 来调试 DragonOS 内核代码了。

您可以参阅 GDB 命令文档来获取更多帮助：GDB Cheat Sheet

CHAPTER 14

内核测试

本章节将介绍如何测试内核，包括手动测试以及自动测试。

我们需要尽可能的对内核进行完善的测试，以便我们能够更好的保证内核的稳定性，且减少其他模块的 debug 难度。

设置完善的测试用例能帮助我们尽可能的检测到问题，防止我们在写新的模块的时候，被已有的模块的一些藏得很深的 bug “背刺一刀”。

由于您难以借助 GDB 等工具进行调试，因此在内核中进行手动测试比应用程序测试要困难一些。

对于一些模块，我们可以使用编写代码进行单元测试，并输出异常信息。遗憾的是，并非所有模块都可以进行单元测试。比如我们常见的内存管理、进程管理等模块都不能进行单元测试。

CHAPTER 15

处理器架构

该部分文档提供了和处理器架构相关的一些编程实现细节的描述。

15.1 x86-64 相关文档

15.1.1 USB Legacy 支持

简介

usb legacy support 指的是，由 BIOS 提供的，对 USB 鼠标、USB 键盘的支持。在支持并启用 USB Legacy Support 的计算机上，USB 鼠标、键盘由 BIOS 提供模拟，在操作系统看来，就像接入了 PS/2 鼠标、键盘一样。

相关

- 在初始化 USB 控制器时，需要关闭它的 USB Legacy Support

CHAPTER 16

其他内核库

这里的集中了内核中的一些库的文档，这些库不属于任何子系统。

16.1 屏幕管理器（SCM）

备注： 作者: 周瀚杰 2625553453@qq.com

屏幕管理器用来管理控制所有 ui 框架，所有框架都必须先在屏幕管理器中注册才可使用，然后 scm 控制当前是哪个 ui 框架在使用

16.1.1 traits

ScmUiFramework

每个要注册到 scm 中的 ui 框架都必须实现这个 trait 中的方法，具体定义如下：

```
pub trait ScmUiFramework: Sync + Send + Debug {
    // 安装ui框架的回调函数
    fn install(&self) -> Result<i32, SystemError> {
        return Err(SystemError::ENOSYS);
    }
    // 卸载ui框架的回调函数
```

(下页继续)

(续上页)

```
    fn uninstall(&self) -> Result<i32, SystemError> {
        return Err(SystemError::ENOSYS);
    }
    // 启用ui框架的回调函数
    fn enable(&self) -> Result<i32, SystemError> {
        return Err(SystemError::ENOSYS);
    }
    // 禁用ui框架的回调函数
    fn disable(&self) -> Result<i32, SystemError> {
        return Err(SystemError::ENOSYS);
    }
    // 改变ui框架的帧缓冲区的回调函数
    fn change(&self, _buf: ScmBufferInfo) -> Result<i32, SystemError> {
        return Err(SystemError::ENOSYS);
    }
    /// @brief 获取ScmUiFramework的元数据
    /// @return 成功: Ok(ScmUiFramework的元数据)
    ///         失败: Err(错误码)
    fn metadata(&self) -> Result<ScmUiFrameworkMetadata, SystemError> {
        // 若文件系统没有实现此方法，则返回“不支持”
        return Err(SystemError::ENOSYS);
    }
}
```

16.1.2 主要 API

scm_init() -初始化屏幕管理模块

原型

```
pub extern "C" fn scm_init()
```

说明

scm_init() 主要是初始化一些 scm 中使用的全局变量，例如是否使用双缓冲区标志位，textui 未初始化时使用的一些全局变量

scm_reinit() -当内存管理单元被初始化之后，重新初始化屏幕管理模块

原型

```
pub extern "C" fn scm_reinit() -> i32
```

说明

scm_reinit() 用于当内存管理单元被初始化之后，重新处理帧缓冲区问题

scm_enable_double_buffer() -允许双缓冲区

原型

```
pub extern "C" fn scm_enable_double_buffer() -> i32
```

说明

scm_enable_double_buffer() 用于启动双缓冲来往窗口输出打印信息。启用后，往窗口输出的信息会暂时放在一个缓冲区中，然后每次按一定时间将该缓冲区的信息输出到窗口帧缓冲区中，渲染显示到窗口上。

scm_framework_enable () -启用某个 ui 框架，将它的帧缓冲区渲染到屏幕上

原型

```
pub fn scm_framework_enable(framework: Arc<dyn ScmUiFramework>) -> Result<i32,␣
↪SystemError>
```

说明

scm_framework_enable 用于启用某个 ui 框架，将它的帧缓冲区渲染到屏幕上

scm_register () -向屏幕管理器注册 UI 框架

原型

```
pub fn scm_register(framework: Arc<dyn ScmUiFramework>) -> Result<i32, SystemError>
```

说明

scm_register 用于将 ui 框架注册到 scm 中，主要是调用 ui 框架的回调函数以安装 ui 框架，并将其激活

16.2 文本显示框架（textui）

备注： 作者: 周瀚杰 2625553453@qq.com

文本框架主要用于 DragonOS 的文本的窗口渲染显示，往屏幕窗口中输出打印文本信息，往窗口显示文本分成两种情况：一种是当内存管理单元（mm）未被初始化时，不能进行动态内存分配，限制颇多（例如不能使用 vec,mpsc 等），所以直接往窗口的帧缓冲区输出打印信息，不使用虚拟行等复杂结构体；另一种是当内存管理单元（mm）已经初始化，可以进行动态内存分配，便可以使用一些复杂的结构体来处理要打印的文本信息。

16.2.1 主要 API

rs_textui_init() -textui 框架初始化

原型

```
pub extern "C" fn rs_textui_init() -> i32
```

说明

rs_textui_init() 主要是初始化一些 textui 框架要使用到的一些全局变量信息（例如 TEXTUIFRAMEWORK，TEXTUI_PRIVATE_INFO 等），以及将 textui 框架注册到 scm 中。

textui_putchar () -往 textui 框架中的当前使用的窗口打印文本信息

原型

```
pub extern "C" fn rs_textui_putchar(character: u8, fr_color: u32, bk_color: u32) ->␣
↪i32

pub fn textui_putchar(
    character: char,
    fr_color: FontColor,
    bk_color: FontColor,
) -> Result<(), SystemError>
```

说明

textui_putchar() 要处理两种情况：一种是当内存管理单元（mm）未被初始化时，不能进行动态内存分配，限制颇多（例如不能使用 vec,mpsc 等），所以直接往窗口的帧缓冲区输出打印信息，不使用虚拟行等复杂结构体；另一种是当内存管理单元（mm）已经初始化，可以进行动态内存分配，便可以使用一些复杂的结构体来处理要打印的文本信息。

16.3 unified-init 统一初始化库

备注： 本文作者：龙进 longjin@DragonOS.org

2023 年 12 月 25 日

16.3.1 1. 简介

该库位于 `kernel/crates/unified-init` 中. 提供统一初始化宏, 用于将函数注册到统一初始化列表中. 便于统一进行初始化.

需要注意的是，初始化器的数组是 no_mangle 的，因此其命名应当遵守模块 _ 初始化器的规则，防止重名导致意想不到的错误.

16.3.2 2. 用法

```
use system_error::SystemError;
use unified_init::define_unified_initializer_slice;
use unified_init_macros::unified_init;

/// 初始化函数都将会被放到这个列表中
define_unified_initializer_slice!(INITIALIZER_LIST);

#[unified_init(INITIALIZER_LIST)]
fn init1() -> Result<(), SystemError> {
   Ok(())
}

#[unified_init(INITIALIZER_LIST)]
fn init2() -> Result<(), SystemError> {
   Ok(())
}

fn main() {
    assert_eq!(INITIALIZER_LIST.len(), 2);
}
```

16.3.3 3. 开发

需要测试的时候可以在 `main.rs` 写测试代码，然后在当前目录执行 `cargo expand --bin unified-init-expand` 就可以看到把 proc macro 展开后的代码了.

16.3.4 4. Maintainer

龙进 longjin@DragonOS.org

CHAPTER 17

应用程序开发文档

17.1 Rust 应用开发快速入门

17.1.1 编译环境

DragonOS 与 Linux 具有部分二进制兼容性，因此可以使用 Linux 的 Rust 编译器进行编译，但是需要进行一些配置：

您可以参考 DragonOS 的 `tools/bootstrap.sh` 中，`initialize_userland_musl_toolchain()` 函数的实现，进行配置。或者，只要运行一下 bootstrap.sh 就可以了。

主要是因为 DragonOS 还不支持动态链接，但是默认的工具链里面，包含了动态链接解释器相关的代码，因此像脚本内那样，进行替换就能运行。

17.1.2 配置项目

从模板创建

备注： 该功能需要 dadk 0.1.4 及以上版本方能支持

1. 使用 DragonOS 的 tools 目录下的 `bootstrap.sh` 脚本初始化环境
2. 在终端输入 `cargo install cargo-generate`

3. 在终端输入

```
cargo generate --git https://github.com/DragonOS-Community/Rust-App-Template
```

即可创建项目。如果您的网络较慢，请使用镜像站

```
cargo generate --git https://git.mirrors.dragonos.org/DragonOS-Community/Rust-App-
↪Template
```

4. 使用 `cargo run` 来运行项目
5. 在DragonOS的 `user/dadk/config` 目录下，使用 `dadk new` 命令，创建编译配置，安装到DragonOS的/目录下。(在 dadk 的编译命令选项处，请使用 Makefile 里面的 `make install` 配置进行编译、安装)
6. 编译 DragonOS 即可安装

手动配置

如果您需要移植别的库/程序到 DragonOS，请参考模板内的配置。

由于 DragonOS 目前不支持动态链接，因此目前需要在 RUSTFLAGS 里面指定`-C target-feature=+crt-static -C link-arg=-no-pie` 并且需要使用上文提到的工具链 `nightly-2023-08-15-x86_64-unknown-linux_dragonos-gnu`

17.2 为 DragonOS 开发 C/C++ 应用

17.2.1 编译环境

DragonOS 与 Linux 具有部分二进制兼容性，因此可以使用 Linux 的 musl-gcc 进行编译。但是由于 DragonOS 还不支持动态链接，因此要增加编译参数`-static`

比如，您可以使用

```
musl-gcc -static -o hello hello.c
```

来编译一个 hello.c 文件。

在移植现有程序时，可能需要配置 `CFLAGS` 和 `LDFLAGS`，以及 `CPPFLAGS`，以便正确地编译，具体请以实际为准。

CHAPTER 18

系统调用 API

18.1 简介

CHAPTER 19

参与开发

DragonOS 社区欢迎您的加入！学习技术本身固然重要，但是以下这些文档将会帮助您了解 DragonOS 社区需要什么。

阅读这些文档将会帮助您参与到开发之中，并且能让您的代码能更快合并到主线。

19.1 开发流程介绍

作为一名想要参与开发的新人，您可能迫切想要了解如何参与开发，仔细阅读这篇文章将能帮助您了解整个开发流程，以及一些注意事项。

注：本文参考了 Linux 文档中的一些思想、内容，非常感谢 Linux 社区的工作者们的经验！

19.1.1 1. 概要

对于新人而言，参与开发的过程主要包括以下几步：

- **运行 DragonOS**：按照文档：*构建 DragonOS*中的教程，编译 DragonOS，并成功运行。在运行过程中，如果有任何的疑问，欢迎您在交流群或者 BBS 上进行反馈！
- **联系 Maintainer**：您可以通过邮箱longjin@DragonOS.org或者 QQ`184829088` 与龙进取得联系，或者是对应的模块的开发者进行联系（目前您可以通过发行日志上的邮箱与他们建立联系，在将来我们将编写一份“Maintainers List”以便于您能快速找到要联系的人）。为了节省您的时间，请简要说明：
  - 如何称呼您

  - 您目前掌握的技术能力
  - 您希望为 DragonOS 贡献什么功能，或者进行某个方面的开发，亦或者是想要按照社区的需要来参与新功能开发及 bug 的修复。
  - 如果您是来自高校/科研单位/企业/组织的代表，希望与社区开展合作研究、开发。那么，除使用 QQ 交流之外，还请麻烦您通过您的教师邮箱/学生邮箱/企业邮箱向contact@DragonOS.org发送一封相关内容的邮件！这么做的目的是为了确认您是来自您的单位，而不是网络上其他人员冒充您的身份。

- **加入工作群**：在进一步了解，确认您愿意参与开发后，我们将邀请您加入工作群。
- **开始开发**：正式开始代码开发工作。

备注： 一些小功能的改进以及 Bug 修复并不一定需要提前与社区建立正式的联系，对于这些 patch，您可以直接开发，然后在 Github 上进行 Pull Request. 这也是可以的。

但是，如果您愿意的话，与 Maintainer 联系会是一个更好的选择。

19.1.2 2. 开发过程是如何运作的？

如今的 DragonOS 由于正处于开发的早期阶段，开发者数量不超过 50 人，因此，现在 DragonOS 的开发过程是通过比较松散的方式组织起来的。

2.1. 版本发布周期

自从 2022 年 11 月 6 日，DragonOS 发布第一个版本以来，版本发布就被定义为 15~21 天发布一个更新版本。由于开发人员数量仍然较少，因此，目前这个时间是 21 天。我们将版本号定义为 `x.y.z` 的格式，每 21 天发布一个 `z` 版本. 在积累了 2 ~ 3 个月后，当 DragonOS 引入了足够的新功能，则发布一个 `y` 版本。请注意，我们仍未定义 `x` 版本的发行周期。当前，`x` 版本号仍然是 `0`。

创建没有 BUG 的、具有尽可能少 BUG 的代码，是每个开发者的目标之一。我们希望在每个 `y` 版本发布时，能够修复已知的问题。然而，由于在操作系统中，影响问题的变量太多了，以至于我们只能尽全力去减少 BUG，我们无法保证 `y` 版本不存在 bug.

2.2. 每个补丁的生命周期

当您向 DragonOS 的仓库发起一次 PR，那么这次 PR 就是一个补丁。我们会对您的补丁进行 Review，以确保每个补丁都实现了一个希望在主线中进行的更改。并且，Maintainer 或者感兴趣的小伙伴会对您的补丁提出修改意见。当时机合适时，您的代码将被合入主线。

如果您的补丁的规模比较小，那么，它将会比较快的被合入主线。如果补丁的规模较大，或者存在一些争议，那么我们需要对其进行进一步的讨论及修改、审查，直到它符合要求。

每个 Patch 都会经历这么一个过程（这只是一个概述，详细的描述请看后文）：

- **设计**：在这个阶段，我们将明确，这个补丁将要实现什么功能，或者是解决什么问题，以及我们将要采用什么样的方式去完成它。通常来说，这项工作是开发者自己完成的。但是，**我们建议您，在设计了这个补丁之后，能够把您的设计方案公开，和大家讨论这项工作**。闭门造车容易出错，在与大家沟通的过程中，则能及早的发现设计上的错误，从而节省很多的时间。
- **代码编写**：经过了设计阶段，您已经能够明白自己要实现的到底是一个什么样的东西。您在这个阶段进行代码编写、调试。
- **代码审查**：当完成代码编写后，您可以通过 Github 发起一个 PR，然后您的补丁进入了代码审查阶段。在这一阶段，开发者们，或者是 Maintainer 会和您进行沟通交流，对您的代码进行评论，以及对发现的问题提出修改建议。
- **合并主线**：如果一切顺利，那么代码将会被合并入主线。若该补丁在合并主线后，被发现具有较大的问题，那么它可能会被回退。重新进入前面的阶段，进行修改。
- **长期维护**：虽然说，代码被合并之后，原来的开发人员可能会在很久一段时间后，不再理会这些代码，但是这种行为可能会给其他开发者们留下不好的印象。其实，当补丁被合并入主线后，其他开发人员会尝试继续维护这些代码，这样能够很大程度的减轻您的维护负担。但是，如果想要这些代码能够长期的被保留下来，持续的发光发热，那么离不开原始开发人员的支持（不然的话，后来的人可能难以理解、维护这些代码），这是一件很有意义的事情。

对于没有参与过开源项目的同学来说，他们可能会想当然的，简单的把上述流程简化成**合并主线**这一个步骤，这样是不可取的。因为这样会导致很多问题，包括但不限于“写了代码但是效果很差”、“写的东西由于无法满足项目的需求，而不能被合并”。

2.3. 开发工具

从上面的描述可以看出，为了让开发人员们能高效地进行协作，那么必须使用版本控制工具来管理这个过程。目前，DragonOS 使用 Git 来进行源代码管理。它是一个非常优秀的工具，这是不必多说的。对于每个开发者而言，Git 的使用是一项必备的技能；哪怕您只是想学习 DragonOS 的源代码，您也需要 git 来获取、同步最新的代码。虽然 Git 的使用，对于新手来说，有些困难，但是经过一些时间的学习后，还是可以掌握的。

git 的官方网站为https://git-scm.com/

2.4. 沟通交流

DragonOS 的主要开发工作是通过飞书以及 bbs 进行的。对于正准备参与的开发者而言，您可以加入我们的交流讨论 QQ 群，具体的群号可以在与社区建立联系 一文中找到。

何时使用即时通讯软件？我们在飞书上创建了工作群，为提高即时沟通的效率，这个群仅供那些真正有意愿、且正在进行或准备进行（能够保证会进行）代码开发的开发者们加入。

何时使用 BBS？对于一些正式的、需要大家广泛参与，或者是能够帮助尚未参与开发的同学了解当前的开发进度的主题，请您在https://bbs.DragonOS.org上，使用类似于写信件一样的，正式的语言，完整地描述清楚您想表达的内容。这样有助于更多的人快速明白您要表达的是什么，也能提高整体的沟通效率。并且，bbs 能够长期保存以往的帖子，这样后来者能更方便的了解“当初开发的时候，人们究竟是怎么想的”。

关于交流讨论会除由于法定节假日放假，或特殊情况以外，我们每周末都会召开线上交流讨论会，同步介绍每周的进展。社区成员可以在这里分享自己的方案设计或是一些操作系统相关的知识（分享的话，需要提前跟会议主持人提出，以便妥善安排）。

如何提问？下面这些建议能够帮助您与他人开展高效率的对话：

- **对于具有主题性的问题，在 bbs 上发帖进行讨论**。这样能够让讨论更具有目标性。当谈论到新的主题的时候，请开一个新的帖子，并在原来的帖子中，添加对特定的子问题所在的主题的链接。
- **请礼貌的交流**。文明的语言能够减少不必要的冲突。技术意见上的冲突是思维的碰撞，但是如果涉及到了不文明的语言，或者在非技术层面，对他人进行攻击，这将破坏和谐讨论的氛围，这是我们反对的。如果有人试图激怒你，请忽略他的消息，别理他就好了。
- **在提问之前，请确保您已经搜索了 bbs 以及互联网上的解决方案，并描述清楚您的问题的上下文情景、您的思考以及网络上的解决方案**。一些开发人员会对“明显没有进行认真思考”的问题，表现出不耐烦的态度（因为未经思考的问题会浪费他们大量的时间）。
- **当别人向您提问时**，请您耐心听他人的问题。如果您认为对方的问题过于简单或是未经思考，还请您为对方指个路，告诉对方，在哪些地方，使用什么样的方式搜索，能够得到对解决问题有帮助的资料。有时候，**新手需要的是一个指路人**，他会非常感谢您的！

2.5. 如何入门开发？

DragonOS 原采用 C 语言进行开发，目前正在用 Rust 重构原有代码、开发新的模块，也就是说，除非您要进行对 C 语言代码的 BUG 修复，否则，其余的开发工作，我们都建议您通过 Rust 完成。因为，它能从语言层面解决那些让我们头疼的内存安全问题。从长期来看，能够提升开发效率以及软件质量。

如何开发第一个补丁，是一个非常常见的问题。可以理解的是，个人开发者面对这样一个项目，常常会不知道从哪个地方开始入手。这是一件很正常的事情，因此我们建议您通过上文提到的方式，与社区建立联系，了解目前社区正在做什么，以及需要什么。

对于一个新的参与者来说，我们建议您从这样一个步骤开始：

```
阅读文档，编译、运行DragonOS，并且尝试使用它目前已有的功能。
```

然后，您可以通过查看 DragonOS 的 GitHub 仓库的 project 面板，看看目前仍有哪些待解决的问题。可以肯定的是，永远不会缺少待解决的问题，您在解决这些问题的过程中，能够获得一些宝贵的经验。

19.1.3 3. 早期设计

对于软件开发而言，写代码永远不是第一步。在编写代码之前，进行一些必要的设计（提出架构、技术方案），是项目成功的基础工作。在新的补丁开发的早期，花费一些时间进行计划和沟通，往往能够在接下来的阶段节省更多的时间。

3.1. 定义我们要解决的问题

与大多数的工程项目一样，在 DragonOS 中进行开发，首先需要清晰的描述要解决的问题。只有精准的定义了问题，才能找到正确的解决方案。有时候，我们能很轻易的定义问题，比如“编写串口驱动程序，使得它能把屏幕上打印的字符，输出到串口”。

但是，有时候，**我们容易将真正的问题与某个解决方案相混淆**，并且还没意识到这一点。

在 2022 年 10 月时，我发现，在真机调试的时候，需要频繁的拔插硬盘（先连接到电脑，待数据拷贝完毕后，再连接到测试机）。我对这一过程非常的不满，因为很浪费时间。我的直觉想法是：“有没有一种设备，能够一头连接电脑，另一头连接测试机的 SATA 接口。从测试机的角度看来，这是一块硬盘；测试机对这块磁盘的操作，都转换为了对我的电脑上面的一个磁盘镜像文件的操作。”我的想法就是：“购买/定制一款设备，能够实现上面的这个功能，那就能解决频繁拔插硬盘的烦恼了！”然后我为了寻找这样的设备，询问了一些硬件公司，他们的开价都在 2 万元左右。

我在上面的这个过程中，就犯了一个错误：将真正的问题与某个解决方案相混淆了。真正的问题是：“解决需要频繁的拔插硬盘”，但是，在我的思考的过程中，不知不觉间，将问题转换成了“如何实现一款具有硬盘模拟功能的设备”。后面这个问题，只是某个解决方案下，需要解决的问题，并不是我们要解决的根本问题。

对于要解决的根本问题，事实上有更好的解决方案：“制作一个类似于开关一样的转换器，当数据从电脑拷贝到磁盘后，把开关拨向另一边，使得电路与测试机接通”。这个方案的成本估摸着就十几二十块钱。

上面的这个故事，告诉我们的是，**在早期设计阶段，我们应当关注的是问题本身——而不是特定的解决方案**。

我们需要关注系统的稳定性、长期的可维护性，解决方案必须考虑到这两点。由于系统比较复杂，因此，请您在开始编码之前，与社区的小伙伴讨论您的设计方案，以便您的方案能充分地，从全局的角度，考虑到系统的稳定性、可维护性。

因此，在开发的早期，我们应当对以下三个问题，拥有一个答案：

- 要解决的本质问题是什么？
- 这个问题会影响哪些方面/哪些用户？提出的解决方案应当解决哪些用例、场景？
- DragonOS 目前在解决该问题的方面，具有哪些不足/问题？

只有考虑清楚了上面三个问题，讨论的解决方案才是有意义的。这是一个架构设计的过程，需要进行仔细的思考。尽管我们目前提倡敏捷开发，但是前期的架构设计仍然是非常重要的。

3.2. 早期讨论

在实施开发之前，与社区的成员们进行讨论是非常有意义的。这能够通过多种方式节省您的时间，并减少许多不必要的麻烦：

- DragonOS 可能以您不知道、不理解的方式，已经解决了相关的问题。DragonOS 里面的一些特性、功能细节不是很明显，他们不一定会被写进文档。也许这些细节只是在某个不起眼的注释里面提到了，因此您很难注意到这些。这种细节可能只有一些开发人员知道。因此，与社区沟通能够避免您进行重复的工作。
- 您提出的解决方案中，可能会有一些东西，由于一些原因（比如方案中的一些设计会在将来造成问题、方案的架构设计具有明显缺陷），无法合入主线。
- 其他的开发人员可能已经考虑过这个问题；他们可能有更好的解决方案，或者是更好的想法。并且，他们可能愿意帮助你一起完善你的解决方案。

Linux 文档中提到：闭门造车式的设计和开发，所产生的代码总会有问题，这些问题只有在发布到社区里面的时候才会暴露出来。因此，我们必须吸取前人之鉴，通过与社区开发人员进行早期讨论，从而避免大量的痛苦和额外的工作。

3.3. 在何时发布帖子？

如果可能的话，在开发的早期阶段发布您的计划、设计，会是一个不错的选择。发帖的时候，您可以描述您正在解决的问题，以及已经制定的一些计划。包括但不限于：如何将设计付诸实施。您在社区内发布帖子，不仅能够帮助您获得一些有用的建议，也能帮助整个 DragonOS 社区提供有用的信息，使得社区沟通更高效。

在这个阶段，可能您发布的帖子并不会引来很多评论，这并不一定意味着您做的不好，或者大家对您所做的工作不感兴趣。当然，也不能就此认为您的设计、想法不存在问题。可能只是因为大家比较忙，看了您的帖子之后，了解到了您的工作，但是大家并不一定有时间进行回复。（但是事实上您发布的信息对他人来说是有用的）

在这种情况下，请不要气馁，您最好的做法就是，继续您的工作，并且不时地在您的帖子下分享您的工作，这样能够让社区的成员们随时了解到您的最新进展。

3.4. 获得您所在的组织的支持

如果您对 DragonOS 的开发工作，是在您的公司内完成的。那么，很显然，在您把计划、代码发布到社区论坛之前，您必须取得您的经理或上级的许可。

同时，请注意，根据我们的授权许可，基于 DragonOS 操作系统的内核、官方开源的用户库而开发的代码，或者为 DragonOS 操作系统本身而开发的代码，根据开源授权许可，必须同样以 GPLv2 协议进行授权发布。如果您所在的组织，违背了 GPLv2 协议中的要求，以除 GPLv2 以外的协议开放源代码，或者是进行“闭源使用”，那么 DragonOS 社区对您的公司/组织所授予的使用 DragonOS 源代码的授权，将会被自动撤销。这将会面临一系列的法律问题。因此，在这个问题上，公司的管理人员、法律人员如果能越早地就公司要在 DragonOS 中开发的软件项目达成一致，将能促进您的公司在该项目上的进展。

如果您的公司的项目/或者是您研究的项目根据您所在组织的保密规定，不能将其进行过早的披露，那也没有问题。只要您的公司能够确保这部分代码，在其编译而成的二进制产品被发布之时，按照 GPLv2 协议进行开源，并向开源社区贡献这部分代码即可。

19.1.4 4. 如何正确的编写代码

19.1.5 5. 发起 Pull Request

19.1.6 6. 后期跟进

19.1.7 7. 另外的一些话题

19.1.8 8. 更多信息

19.1.9 9. 结语

19.2 C 语言代码风格

这份文档将会简要的介绍 DragonOS 的 C 语言代码风格。每个人的代码风格都各不相同，这是一件非常正常的事情。但是，对于一个开源项目的可维护性而言，我们希望制定一些代码规范，以便包括您在内的每个开发者都能在看代码的时候更加舒服。一个充斥着各种不同代码风格的项目，是难以维护的。

我们在这里提出一些建议，希望您能够尽量遵循这些建议。这些建议与 Linux 的代码规范相似，但又略有不同。在变量命名上，DragonOS 采用 Linux 的风格；对于缩进，DragonOS 采用 Microsoft 风格。

19.2.1 0. 代码格式化工具

在提出下面的建议之前，我们建议您在开发的时候使用 Visual Studio Code 的 `C/C++ Extension Pack` 插件作为代码格式化工具。这些插件能为您提供较好自动格式化功能，使得您的代码的基本格式符合 DragonOS 的要求。

当您在编码时，经常性的按下 `Ctrl+shift+I` 或您设置的代码格式化快捷键，能帮助您始终保持良好的代码格式。

19.2.2 1. 缩进

一个制表符的宽度等于 4 个空格。代码的缩进是按照制表符宽度 (在多数编辑器上为 4 个字符) 进行缩进的。

这样能够使得您的代码变得更加容易阅读，也能更好的看出代码的控制结构。这样能避免很多不必要的麻烦！

举个例子：在 switch 语句中，将 switch 和 case 放置在同一缩进级别。并且将每个 case 的代码往右推进一个 tab。这样能让代码可读性变得更好。

```
switch (cmd)
{
case AHCI_CMD_READ_DMA_EXT:
    pack->blk_pak.end_handler = NULL;
    pack->blk_pak.cmd = AHCI_CMD_READ_DMA_EXT;
    break;
case AHCI_CMD_WRITE_DMA_EXT:
    pack->blk_pak.end_handler = NULL;
    pack->blk_pak.cmd = AHCI_CMD_WRITE_DMA_EXT;
    break;
default:
    pack->blk_pak.end_handler = NULL;
    pack->blk_pak.cmd = cmd;
    break;
}
```

19.2.3 2. 分行

我们建议，每行不要超过 120 个字符。如果超过了，除非有必要的理由，否则应当将其分为两行。

在分行时，我们需要从被分出来的第二行开始，比第一行的起始部分向右进行一个缩进，以表明这是一个子行。使用代码格式化的快捷键能让你快速完成这件事。

对于一些日志字符串而言，为了能方便的检索到他们，我们不建议对其进行分行。

对于代码的分行，请不要试图通过以下的方式将几个语句放置在同一行中，这样对于代码可读性没有任何好处：

```
// 错误示范(1)
if(a) return 1;

// 错误示范(2)
if(b)
    do_a(),do_b();
```

19.2.4 3. 大括号和空格

3.1 大括号

大括号的放置位置的选择是因人而异的，主要是习惯原因，而不是技术原因。我们推荐将开始括号和结束括号都放置在一个新的行首。如下所示：

```
while(i<10)
{
    ++i;
}
```

这种规定适用于所有的代码块。

这么选择的原因是，在一些编辑器上，这样放置括号，**编辑器上将会出现辅助的半透明竖线，且竖线两端均为括号**。这样能帮助开发者更好的定位代码块的层次关系。

下面通过一些例子来演示：

在下面这个代码块中，我们需要注意的是，`else if` 语句需要另起一行，而不是跟在上一个`}`后方。这是因为我们规定 `{`必须在每行的起始位置，并且还要保持缩进级别的缘故。

```
if (*fmt == '*')
{
    ++fmt;
}
else if (is_digit(*fmt))
{
    field_width = skip_and_atoi(&fmt);
}
```

当循环中有多个简单的语句的时候，需要使用大括号。

```
while (condition)
{
    if (test)
        do_something();
}
```

当语句只有 1 个简单的子句时，我们不必使用大括号。

```
if(a)
    return 1;
```

3.2 空格

对于大部分关键字，我们需要在其后添加空格，以提高代码的可读性。

请您在所有这些关键字后面输入一个空格：

```
if, switch, case, for, do, while
```

关键字 sizeof、typeof、alignof、__atrribute__ 的后面则不需要添加空格，因为使用他们的时候，就像是使用函数一样。

对于指针类型的变量，* 号要贴近变量名而不是贴近类型名。如下所示：

```
char *a;
void *func(char* s, int **p);
```

在大多数二元和三元运算符周围（在每一侧）使用一个空格，如下所示：

```
=  +  -  <  >  *  /  %  |  &  ^  <=  >=  ==  !=  ?  :
```

这些一元运算符后方没有空格

```
&  *  +  -  ~  !  sizeof  typeof  alignof  __attribute__  defined
```

特殊的例子，以下运算符的前后都不需要空格：

```
++  --  .  ->
```

19.2.5 4. 命名

DragonOS 中的命名规范不使用诸如 `TempValue` 这样的驼峰命名法的函数名，而是使用 `tmp` 这样言简意赅的命名。

注意，这里指的是我们在整个项目内都不希望使用驼峰命名法。并不意味着程序员可以随便的使用一些难以理解的缩写来作为变量名。

对于全局变量或者全局可见的函数、结构体而言，我们需要遵循以下的命名规定：

- 名称需要易于理解，且不具有歧义。如：对于一个计算文件夹大小的函数而言，我们建议使用 `count_folder_size()` 来命名，而不是 `cntfs()` 这样令其他人头大的命名。
- 全局的，非 static 的名称，除非有特别的必要，命名时需要遵循以下格式：*模块名缩写前缀* `_` *函数/变量名*。这样的命名能便于别人区分这个名称位于哪个模块内，也减少了由于命名冲突所导致的麻烦。
- 不需要让其他代码文件可见的全局名称，必须添加 `static` 修饰符。

对于函数内的局部变量而言，命名规范则是需要言简意赅。过长的名称在局部变量中没有太大的意义。

【文档未完成，待继续完善】

19.3 Rust 语言代码风格

这篇文档将会介绍 DragonOS 中的 Rust 语言代码风格。随着开发的进行，这些风格可能会发生变化，但是我们会尽量保持风格的一致性。

19.3.1 1. 命名

这部分基于 Rust 语言圣经中的命名规范进行修改，本文未提及的部分，请参考 Rust 语言圣经中的命名规范。

19.3.2 2. 格式

2.1 缩进

请在提交代码之前，使用 `cargo fmt` 命令对代码进行格式化。

2.2 函数返回值

尽管 Rust 可以返回函数的最后一行的语句的值，但是，这种方式会使代码的可读性变差。因此，我们推荐您在函数的最后一行使用 `return` 语句，而不是直接返回值。

```
// 不推荐
fn foo() -> i32 {
    1 + 2
}

// 推荐
fn foo() -> i32 {
    return 1 + 2;
}
```

2.3 错误处理

DragonOS 采用返回 Posix 错误码作为**模块间错误处理**的方式。为了确保在模块之间，错误处理代码的一致性，我们推荐在发生错误的时候，返回 `SystemError` 类型，该类型表示 posix 错误码。这样做的优点尤其体现在跨模块调用函数时，可以直接返回通用的错误码，从而降低错误处理代码的耦合度。

```
// 函数跨越模块边界时（由其他模块调用当前函数），不推荐
fn foo() -> Result<(), CustomErr> {
    if 1 + 2 == 3 {
        return Ok(());
```

(下页继续)

(续上页)

```
    } else {
        return Err(CustomErr::error);
    }
}

// 函数跨越模块边界时（由其他模块调用当前函数），推荐
fn foo() -> Result<(), SystemError> {
    if 1 + 2 == 3 {
        return Ok(());
    } else {
        return Err(SystemError::EINVAL);
    }
}
```

在**模块内部**，您既可以采用返回自定义错误 enum 的方式，也可以采用返回 `SystemError` 的方式。但是，我们推荐您在模块内部，采用返回自定义错误 enum 的方式，这样可以使错误处理代码更加清晰。

TODO: 将原有的使用 i32 作为错误码的代码，改为使用 `SystemError`。

19.3.3 3. 注释

DragonOS 的注释风格与 Rust 官方的保持一致。同时，我们推荐您在代码中加入尽可能多的有效注释，以便于其他人理解您的代码。并且，变量、函数等声明，遵守第一节中提到的命名规范，使其能够“自注释”。

3.1 函数注释

函数注释应该包含以下内容：

- 函数的功能
- 函数的参数
- 函数的返回值
- 函数的错误处理
- 函数的副作用或者其他的需要说明的内容

函数注释的格式如下：

```
/// # 函数的功能
///
/// 函数的详细描述
///
/// ## 参数
///
```

(下页继续)

(续上页)

```
/// - 参数1: 参数1的说明
/// - 参数2: 参数2的说明
/// - ...
///
/// ## 返回值
/// - Ok(返回值类型): 返回值的说明
/// - Err(错误值类型): 错误的说明
///
/// ## Safety
///
/// 函数的安全性说明
```

19.4 代码提交规范

这份文档将会简要的介绍 DragonOS Github 仓库的代码提交规范，主要是给出了基于 Conventional Commit 的命名规范，以及对 DragonOS Bot 的简要介绍

19.4.1 Conventional Commit(约定式提交)

关于约定式提交的详细规范，在Conventional Commit/约定式提交网站中有详细介绍，在本节末尾将给出示例 (摘自Conventional Commit/约定式提交网站)，可选择性阅读。我们做出以下特别说明：

1. 由于 DragonOS 内核原则上仅通过系统调用接口保证对外可用性，而迄今为止 (2024/04/22)，出于对软件生态的考量，DragonOS 选择实现与 Linux 一致的系统调用，因此不会对破坏性变更 (BREAKING CHANGES) 做特殊说明，或者说，在当前开发环境中不会产生对用户产生显著影响的破坏性变更，因此无特殊需要，DragonOS 内核不应使用诸如 `feat!` 来表示破坏性变更。(内核之外，例如 dadk，仍需遵循规范)

2. DragonOS 社区严格遵循基于 squash 的工作流，因此我们并不强制要求 PR 中的每一个单独的 commit 都符合Conventional Commit/约定式提交，但是我们仍强烈建议使用。

3. 关于 scope: 如无特殊说明，以子模块/系统/目录名作为范围，例如代码变动是发生在 `kernel/src/driver/net` 中的特性追加，那么应当命名为 `feat(driver/net):`，如果是发生在 `kernel/src/mm/allocator` 中，应当命名为 `feat(mm)`，简而言之就是要尽可能简短的表现出所属模块，大多数情况下，不应使用超过两级的范围标识，例如 `fix(x86_64/driver/apic)` 是错误的，应当命名为 `fix(x86_64/apic)`

4. 在 DragonOS 内核代码仓库中的 `issue checker` 会对标题格式进行简单审查，如果不符合格式的将会被标记为 `ambiguous`，贡献者们请按需修改

5. 使用小写

示例

包含了描述并且脚注中有破坏性变更的提交说明

```
feat: allow provided config object to extend other configs

BREAKING CHANGE: `extends` key in config file is now used for extending other config␣
↪files
```

包含了! 字符以提醒注意破坏性变更的提交说明

```
feat!: send an email to the customer when a product is shipped
```

包含了范围和破坏性变更! 的提交🄵明

```
feat(api)!: send an email to the customer when a product is shipped
```

包含了! 和 BREAKING CHANGE 脚注的提交说明

```
chore!: drop support for Node 6

BREAKING CHANGE: use JavaScript features not available in Node 6.
```

不包含正文的提交说明

```
docs: correct spelling of CHANGELOG
```

包含范围的提交说明

```
feat(lang): add polish language
```

19.4.2 DragonOS Bot

DragonOS 使用 triagebot 来实现自动标签功能以及分配 reviewer，贡献者也可以通过部分命令与 triagebot 交互，详见triagebot

CHAPTER 20

与社区建立联系

20.1 联系方式

社区公共邮箱：contact@DragonOS.org

DragonOS 社区负责人: 龙进

工作邮箱: longjin@DragonOS.org

开发交流 QQ 群：115763565

DragonOS 官网：https://DragonOS.org

了解开发动态、开发任务，请访问 DragonOS 的论坛：https://bbs.dragonos.org.cn

20.2 赞助及捐赠

DragonOS 是一个开源项目，我们欢迎任何形式的赞助和捐赠，您的捐赠将用于 DragonOS 的开发和维护，以及社区的运营。

您可以通过以下方式赞助和捐赠：

- 访问 DragonOS 官网 https://DragonOS.org ，点击页面右上角的“赞助”按钮，进行捐赠
- 联系社区负责人，沟通具体的赞助方式等。联系方式：longjin@dragonos.org

20.3 财务及捐赠信息公开

DragonOS 社区的捐赠信息将按年进行公开。赞助商、赞助者信息将在收到赞助后，15 天内进行公开。

20.4 社区管理、财务及法务主体

DragonOS 社区的管理、财务及法务主体为：灵高计算机系统（广州）有限公司。

我们是一家开源公司，我们坚信，开源能为我国将来的 IT，打下更好的基础。我们也通过其他业务创收，投入到 DragonOS 的研发之中。

公司负责 DragonOS 社区的运营、财务、法务事项处理工作。

地址：广东省广州市番禺区小谷围街广州大学城华南理工大学大学城校区

邮件：contact@DragonOS.org

官网：https://ringotek.com.cn

CHAPTER 21

发行日志

这里是 DragonOS 的发行日志，会记录 DragonOS 的每一个版本的更新内容。

21.1 V0.1.9

备注： 本文作者：何懿聪 heyicong@dragonos.org

DragonOS 官方论坛：bbs.dragonos.org.cn

2024 年 3 月 13 日

21.1.1 贡献者名单

DragonOS V0.1.9 版本由以下小伙伴贡献代码：

- 龙进 longjin@DragonOS.org
- 何懿聪 heyicong@dragonos.rog
- 裕依 68320855+yuyi2439@users.noreply.github.com
- R0ronoa 84278015+2447742618@users.noreply.github.com
- 池克俭 39649411+Chiichen@users.noreply.github.com
- 吴宇健 wuyujian@dragonos.org

- zhaoyao73 zhaoyao73@users.noreply.github.com
- 胡兆朋 105195940+MemoryShore@users.noreply.github.com
- 周瀚杰 zhouhanjie@dragonos.org
- 栗子 im.lechain@gmail.com
- Xshine gshine@m.scnu.edu.cn
- Chenzx 109664121+schulice@users.noreply.github.com
- MContour m-contour@qq.com
- Donkey Kane 109840258+xiaolin2004@users.noreply.github.com
- Luo Jia / Zhouqi Jiang luojia@hust.edu.cn
- Wu Mianzhi 31810920+Hdksg10@users.noreply.github.com
- Xiaoye Zheng xiaoyez@zju.edu.cn
- Plucky923 107762234+Plucky923@users.noreply.github.com
- 许梓毫 xuzihao@dragonos.org

21.1.2 赞助商列表

- **中国雅云** 雅安大数据产业园为 DragonOS 提供了云服务器支持。

21.1.3 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- David Wen
- 万晓兰
- 龙进
- 吴宇健

两千元以下：

- Seele.Clover
- FindWangHao
- ferchiel
- 叶锦毅
- 林
- Albert

- TerryLeeSCUT · GitHub
- slientbard
- 悟
- 匿名热心人士

21.1.4 更新内容-内核

新特性

- refactor: 重构进程管理模块 (#380)
- feature: 完善设备驱动模型 (#401)
- feature: 实现 e1000e 网卡驱动 (#393)
- feature: DragonOS 虚拟化框架 (#389)
- feature: 支持 syscall 快速系统调用指令 (#417)
- refactor: 重写 apic 驱动 (#425)
- feature: 线程机制与 futex (#411)
- feature: DragonStub 引导 DragonOS 内核 (#460)
- feature: 实现 Epoll IO 多路复用机制 (#455)
- feature: 帧缓冲抽象以及 vesafb 驱动 (#483)
- feature: 增加 early io remap 的 fixmap 功能 (#495)
- feature: 实现内核日志系统 (#489)
- refactor: 使用 Rust 重写 x86_64 下内核初始化代码 (#507)
- feature: 新增 riscv64 架构的内存管理等 (#506)
- refactor: 中断管理模块重构完成 (#554)
- refactor: 重构 tty 模块，实现 unix 兼容 tty(#577)
- featurn: 实现若干 POSIX 标准系统调用

bugfix

- bugfix: 修正由于 init proc union 导致的无法运行的问题 && 修正由于内核线程启动后默认 sleep 的行为导致 init 进程无法正常运行的 bug (#381)
- bugfix: 修复了 Flusher Drop 的时候没有自动刷新 TLB 的 bug(#384)
- bugfix: multiboot2 启动的信息因为没及时转存导致后面无法从其中进行查询的 bug (#405)
- bugfix: 修复 bus/device manager 对卸载逻辑的处理错误 (#385)
- bugfix: 解决 waitqueue sleep 的时候, 由于 preempt count 不为 0, 导致 sched 失败, 从而导致该 waitqueue 下一次 wakeup 时, 会把 pcb 多次加入调度队列的 bug (#419)
- bugfix: 修正 fork 的时候没有正确拷贝 vm holes 的 bug (#433)
- bugfix: 当物理机具有多个 memory area 的时候, 无法正确使用这些区域的问题. 以及在内核代码处出现内存空洞而导致无法正常运行的问题. (#448)
- bugfix: 修复因 rsdp v1 v2 版本问题，导致 ACPI 无法正常初始化的 bug (#454)
- bugfix: 修正由于 bus 的 driver、device 强弱引用关系不正确从而导致对象被释放的 bug (#483)
- bugfix: 修复文件关闭后 epoll 还持有对应描述符的文件弱引用的 bug (#455)
- bugfix: 修复无法 sleep 的问题以及进程处于 block(true) 状态时无法被信号唤醒 & 唤醒后不处理信号的问题 (#470)

21.1.5 更新内容-用户环境

新特性

- feature: 新增 init 程序 dragonreach (#391)
- featurn: 新增 shell 程序 NovaShell (#456)
- featurn: 新增文本编辑器 Held (#583)
- featurn: 能够支持 gcc，tar，redis 等程序运行

21.1.6 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/DragonOS-Community/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org.cn
- https://git.mirrors.DragonOS.org.cn

21.1.7 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了 Linux 社区的一些设计，或者引入了他们的部分思想，亦或是受到了他们的启发。我们在这里对 Linux 社区以及 Linux 社区的贡献者们致以最衷心的感谢！

21.1.8 当前版本的所有提交记录

```
commit af59116e1b8643862598607dbc6ef7233f3791b5
Author: MemoryShore <105195940+MemoryShore@users.noreply.github.com>
Date:   Tue Mar 12 17:52:14 2024 +0800

    Update NovaShell version to c6454d3220 (#593)

commit 59fdb447ee4f7b53b1d9c56ec1442aa8c597ac2b
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Mar 12 16:32:33 2024 +0800

    fix: 键盘中断上下文不再直接操作tty,而是由专门的kthread来渲染 (#592)

    fix: 键盘中断上下文不再直接操作tty,而是由专门的kthread来渲染
    1.修正psmouse 日志
    2. 键盘中断上下文不再直接操作tty,而是由专门的kthread来渲染
    3. 由于调度器设计问题,load balance会由于时序问题导致错误.因此暂时只启用单核.

commit 818a64c77613a9c2152739f1cddad78d61e4a94f
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Mar 12 15:33:01 2024 +0800

    暂时禁用load balance (#591)

    原因见issue: https://github.com/DragonOS-Community/DragonOS/issues/571

commit 4374bd1d1177dbf94112aee3ea3f8e8c335a599c
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Mar 11 19:40:52 2024 +0800

    修复get_random一个问题，添加Held配置文件 (#583)

commit 52bcb59e9286def2b66d766f6bf6f46745795ec8
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Mar 11 15:13:37 2024 +0800

    完善Tty的RawMode (#577)
```

(下页继续)

(续上页)

```
    * 完善rowmode，改掉一部分bug

    * 增加两个ansi拓展功能功能，以及标记部分函数nerve inline

    * 修改do_signal和其他中断上下文锁未关中断，以及拓展tty功能，修改tty几个算法bug

    * 修改两个锁

    * 修改syscall_64

    * update

commit 840045af94ea3391f29e87e968db5d9c48316981
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Mar 10 21:45:34 2024 +0800

    引入clippy，并根据clippy的提示，修改部分代码 (#575)

commit f4a82aa55c55e1a9233e99a5598e180f0858d877
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Mar 10 20:42:41 2024 +0800

    Update Novashell version to 473d5c403c (#574)

    - fix: 用户输入不正确的;以及单独输入单引号和双引号造成系统重启
    - 修改命令解析算法
    - fix: 输出多余的光标和命令信息

commit 4f8f484930ed3c09ecf4b5b05b1dea14f7b05d8b
Author: 栗子 <im.lechain@gmail.com>
Date:   Sat Mar 9 21:20:12 2024 +0800

    修复Archlinux下的bootstrap脚本问题， (#552)

    * 修复Archlinux下的bootstrap脚本问题，

    由于archlinux 的 texinfo版本太新导致的gcc docs构建失败
    使用 MAKEINFO=true 的make环境变量跳过gcc docs构建，绕过问题


    ---------
```

(下页继续)

(续上页)

```
    Co-authored-by: longjin <longjin@dragonos.org>

commit 3055390c25bb7b12279df174689ba09ec50c7d46
Author: Jomo <xuzihao@dragonos.org>
Date:   Sat Mar 9 11:40:44 2024 +0800

    完善重映射过程中获取新映射区域时的map_flags (#569)

commit 5c4224e5a8244cb0fb32512e70354362fccd6321
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Mar 8 23:23:06 2024 +0800

    在riscv上实现异常处理，能够进入异常处理程序 (#564)

commit c3dc6f2ff9169c309d1cbf47dcb9e4528d509b2f
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Mar 8 23:22:48 2024 +0800

    删除一些过时的C代码 (#565)

    * 删除C版本的crc库

    * 删除lockref

    * 删除过时的libc文档以及wait.c

    * 删除过时的C版本kfifo代码及文档

    * 移除未用到的lz4库

    * 删除内核的stdlib.c

    * 删除fabs.c

    * fmt

    * 使得put_string系统调用能够通过tty输出颜色
    暂且这样改，这一部分应该是用户层面提供的功能，后续删除

    ---------

    Co-authored-by: GnoCiYeH <heyicong@dragonos.org>
```

(下页继续)

(续上页)

```
commit 5eeefb8c80e5580641d295724f8d2190bd54979c
Author: Chenzx <109664121+schulice@users.noreply.github.com>
Date:   Fri Mar 8 16:01:22 2024 +0800

    实现SYS_RMDIR (#566)

    * 实现rmdir系统调用，整理do_remove_dir逻辑

commit 338f6903262c5031abad3c8e361813355a27fcdb
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Mar 5 17:22:04 2024 +0800

    `riscv`: 初始化irq (#560)

    完成riscv的irqchip初始化的代码。

    这是该功能的第一个PR。由于还需要实现timer驱动才能测试，因此该功能将会通过2~
↪3个PR来完成。

commit bc6f0a967c8cb1e9379ced184b25a7722fbda2a4
Author: 裕依 <68320855+yuyi2439@users.noreply.github.com>
Date:   Mon Mar 4 14:20:01 2024 +0800

    移除relibc和old libc以及旧的shell (#529)

    移除relibc和old libc以及旧的shell

commit f3b05a97ec061e766247b18dc12e2a413b977b14
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Mar 4 14:10:26 2024 +0800

    将tty输出接入串口 (#555)

commit cf45dede2e46d9341cec75871fdc1bc5961ba5a2
Author: MContour <m-contour@qq.com>
Date:   Sun Mar 3 17:20:11 2024 +0800

    让 DragonOS 仓库管理 service 文件 (#548)

    * feat: create `user/services` folder to store service files

commit e28411791f090c421fe4b6fa5956fb1bd362a8d9
Author: LoGin <longjin@DragonOS.org>
```

(下页继续)

(续上页)

```
Date:   Sun Mar 3 16:31:08 2024 +0800

    完成中断管理模块重构 (#554)

    - 支持中断共享
    - 把现有驱动程序移植到新的irq模块
    - 使用`ProcessorId`标识处理器id
    - 尚未实现threaded_irq

    性能上，edge irq flow handler里面，对于锁的使用，可能有点问题。为了获取/
↪修改common␣
↪data还有其他几个结构体的状态，进行了多次加锁和放锁，导致性能降低。这是接下来需要优化的点。

commit 44d051e5864adff6f4ace8a234ef57852840f365
Author: Donkey Kane <109840258+xiaolin2004@users.noreply.github.com>
Date:   Sun Mar 3 15:22:45 2024 +0800

    Modify dadk config to switch NovaShell revision (#550)

commit 0e7c46939604a02e739546200bea847f4951a963
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Fri Mar 1 15:07:00 2024 +0800

    修改事件等待队列对于retain使用方法出错导致唤醒错误 (#551)

commit be60c929c8285d3050e022aa23312a84129e54b2
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Wed Feb 28 20:18:49 2024 +0800

    修改tty几个bug (#549)

    * 更改ioctl一处逻辑错误

    * 删除不必要的impl

    * 修改一处bug，并且加入tty的link，为pty做准备

    * 修改一处因为vc的pos和x计算错误导致的溢出

commit 52da9a59374752b4d01907b052135a0d317781dd
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Feb 26 15:27:19 2024 +0800
```

(下页继续)

(续上页)

```
    完成与Linux兼容的Ntty (#517)

    * 已经完成的功能：
    - 写：printf能够正常在tty输出
    - 读：与键盘驱动接上
    - 信号： 能够正常通过ctrl向前台进程发送信号

    * 支持目前的shell，改动printk使其与新版tty兼容。

    * 删除原有tty文件夹，并更改新tty文件名

    * 添加clear清屏程序

    * 实现tty部分ioctl，更改部分问题

commit 9993c0fc61e9603f631bd6748ff0b4fecb7bd483
Author: R0ronoa <84278015+2447742618@users.noreply.github.com>
Date:   Mon Feb 26 15:03:13 2024 +0800

    添加i8042驱动 改正serio设备文件结构 (#538)

    * 添加i8042驱动 改正serio设备文件结构

commit d2b28acb4d1f160779b25d76afca49ed60ad5d48
Author: 栗子 <im.lechain@gmail.com>
Date:   Sun Feb 25 19:57:30 2024 +0800

    tools/bootstrap.sh: fix archlinux and zsh issue (#535)

    1. archlinux上没有libssl-dev包，相对应的包名是openssl
    2. zsh用户使用bootstrap.sh 会因为 source ~/.zshrc
       导致bash执行很多zsh built-in 指令出现非常多错误
    3. mkdir命令加-p选项避免对象目录存在时（反复执行脚本）报错
    4. 修复简单错误(typo gcc 为 binutils)
    5. 修复bootstrap.sh等脚本无法在非tools/目录执行的错误

commit 7d66c3134c1c5566ac8f8b7524e98650d0480a4a
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 19 21:40:37 2024 +0800

    添加简单的cpumask (#533)

commit b2ca6800f9d943e5d3656d9b50a099da768775a7
```

(下页继续)

(续上页)

```
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 19 19:50:03 2024 +0800

    添加动态申请的bitmap (#532)

commit 0e2c2e8b48369b201f44e0933f775b932b6776ef
Author: 裕依 <68320855+yuyi2439@users.noreply.github.com>
Date:   Mon Feb 19 19:31:17 2024 +0800

    修正systemerror号 (#527)

    修正systemerror号

commit 4cfa009b87c8431a41fab740ffdbd7b008965c9a
Author: Jomo <xuzihao@dragonos.org>
Date:   Mon Feb 19 14:54:11 2024 +0800

    实现mremap系统调用 (#518)

    * mremap系统调用

commit 27b967a38a6dd7a266c43b5e705c29dfbbd71ae4
Author: 裕依 <68320855+yuyi2439@users.noreply.github.com>
Date:   Mon Feb 19 14:53:34 2024 +0800

    添加pread&pwrite (#528)

    添加pread&pwrite

commit 701589559f912deb03eb5176d049d9d07fb29447
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 19 11:17:23 2024 +0800

    删除无用的C版本bitree和ida/idr. (#526)

    这些数据结构不再使用,将其删除.

commit 196b75dc17b5cc2ed84301bce776e496ddfe1ed1
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 19 00:56:58 2024 +0800

    把irqdesc添加到sysfs (#525)
```

(下页继续)

(续上页)

```
    * 把irqdesc添加到sysfs

commit 3bc96fa4a9c01d91cddeb152fe78d6408351c29f
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 19 00:36:36 2024 +0800

    添加irqdesc的抽象，并在系统初始化时创建irqdesc (#522)

    * 添加irqdesc的抽象，并在系统初始化时创建irqdesc

commit ce5850adbf74ec6c6717bbb5b1749f1fbff4ca0d
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Feb 18 20:41:41 2024 +0800

    添加irqchip这一层的数据结构(尚未接入真实的芯片) (#520)

    * 添加irqchip这一层的数据结构(尚未接入真实的芯片)

commit ca318c376bd9e39f8fe71f304974f7e99e8e01f4
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Feb 17 01:51:10 2024 +0800

    update dragonreach to 3d99c3a9d9 (#519)

commit 472f0b3931eadda2bbcc67889d612790f147190b
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Feb 13 12:14:12 2024 +0800

    update dragonreach to 40362c48d6 (#516)

commit d90848514bea7efd5898f5824b98f1bf2e54de8d
Author: Luo Jia / Zhouqi Jiang <luojia@hust.edu.cn>
Date:   Sat Feb 10 23:20:50 2024 +0800

    riscv: 更新sbi-rt至0.0.3版本 (#512)

    先前使用git仓库链接的最新更新已发布至crates.io网站

    Signed-off-by: Zhouqi Jiang <luojia@hust.edu.cn>

commit 4ad52e57e612a88ab09413c7ac0072db96a93632
Author: 裕依2439 <68320855+yuyi2439@users.noreply.github.com>
Date:   Wed Feb 7 18:06:15 2024 +0800
```

(下页继续)

(续上页)

```
    添加socketpair (#505)

    * 添加对socketpair系统调用的处理

    ---------

    Co-authored-by: LoGin <longjin@DragonOS.org>

commit cb23beb255d8e32b45d879ac19386a3597ca4115
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Feb 7 17:15:17 2024 +0800

    riscv: probe sbi extensions (#511)

commit f2022a8a1cc4a8e2a85e9061e036e9c491a2fa00
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Feb 7 13:29:47 2024 +0800

    使用rust编写中断/异常的入口 (#509)

    * 使用rust编写中断/异常的入口

commit d14e28a8a9b023ee8df7c2e8eee43e523134dbb2
Author: Luo Jia / Zhouqi Jiang <me@luojia.cc>
Date:   Wed Feb 7 11:38:15 2024 +0800

    riscv: 使用sbi-rt库完成SBI操作 (#510)

    未来的其它SBI操作也将使用sbi-rt

    Signed-off-by: Zhouqi Jiang <luojia@hust.edu.cn>

commit fccbe87dcae0f8e8fde611ef60b1f7923126d526
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Feb 5 14:50:15 2024 +0800

    删除关于zulip的表述,替换为社区论坛 (#508)

commit 5b59005f930266d0e9c0092373e894826150f862
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Feb 4 15:46:24 2024 +0800
```

(下页继续)

(续上页)

```
    x86_64: 使用Rust重写内核初始化代码 (#507)

    * x86_64: 使用Rust重写内核初始化代码

commit 453452cc02e2766a28d87dd47bdee37caddc4c44
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Feb 4 14:35:18 2024 +0800

    初始化riscv的内存管理模块，并且设置保留内存 (#506)

commit a02ce654cf0166720f8569827d0c5b2dfd0ca95a
Author: Chiichen <chiichen@qq.com>
Date:   Sun Jan 28 20:29:01 2024 +0800

    支持对动态链接文件的加载，支持通过musl工具链编写用户空间程序 (#504)

    支持对动态链接文件的加载，支持通过musl工具链编写用户空间程序

commit 5d549a76ab0cf66651a6614be92dcb481fe7af2a
Author: Chiichen <chiichen@qq.com>
Date:   Fri Jan 26 20:45:08 2024 +0800

    修改一处常量定义 (#503)

    * 修改一处常量定义

    * 修复ELF加载程序对用户程序加载地址判断不正确的地方

    ---------

    Co-authored-by: longjin <longjin@DragonOS.org>

commit 9284987850c1da2ce607a539eeae8a353a8f6df9
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Jan 26 18:08:39 2024 +0800

    riscv: 完成UEFI初始化,能正确设置memblock的信息 (#501)

    * riscv: 完成UEFI初始化,能正确设置memblock的信息

    * sbi增加reset功能

    * 把虚拟CPU修改为sifive-u54，使qemu能更正确地模拟硬件行为
```

(下页继续)

(续上页)

```
    * 修复内存页面映射未设置“DIRTY”、”ACCESSED“、”GLOBAL“位，导致真机page␣
↪fault的问题

commit a381e482cbe742b2e4bbeaacae134a8131bf3f91
Author: R0ronoa <84278015+2447742618@users.noreply.github.com>
Date:   Wed Jan 24 19:17:22 2024 +0800

    实现Ps/2鼠标驱动 (#500)

    实现Ps/2鼠标驱动

commit 8d72b68da9339ec97e1b8929bcf2946f0fd83cd5
Author: Jomo <xuzihao@dragonos.org>
Date:   Wed Jan 24 16:13:15 2024 +0800

    实现内核日志系统 (#489)

    * 实现写日志和读取日志，并且能够在用户态下执行dmesg命令查看日志

    * 通过klogctl实现dmesg

    * 改用ConstGenericRingBuffer作内核缓冲区

    * 更改缓冲区容量

    *␣
↪将能够输出到控制台的日志级别改为日志级别枚举类，使用SpinLock控制KMSG，使用枚举类定义SYSLOG_
↪ACTION，将do_syslog系统调用接口放在syscall.rs

    * fix warning

    * 完善do_syslog注释

    * 将KMSG接入kinfo、kdebug等

    * fix warning

    * 修复显示的秒数不正确，·以及无法通过CI的问题

commit d46c6d27941a26de14f55a2bbf956219bcc70871
Author: 裕依2439 <68320855+yuyi2439@users.noreply.github.com>
Date:   Tue Jan 23 23:36:52 2024 +0800
```

(下页继续)

(续上页)

```
    修复tty的buf满时始终阻塞的问题 (#493)

    * 修复tty的buf满时始终阻塞的问题

commit 43ef2a0d2b6ec427f6775cd4593c56897dd5bf6d
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Jan 21 18:45:07 2024 +0800

    添加riscv64的github workflow (#499)

    * 添加riscv64的github workflow

commit 7a29d4fcbcd89a226289c7bf541c2c78623de3ad
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Jan 21 15:38:12 2024 +0800

    riscv64: 映射uefi systemtable,并完善了riscv64页表填写的部分内容 (#498)

    * 从fdt的chosen段获取几个需要的字段

    * merge patch-early-ioremap

    * feature: 增加early io remap的fixmap功能

    允许在内存管理初始化之前,使用fixmap功能,映射一些物理内存,并记录.

    * riscv64: 映射uefi systemtable,并完善了riscv64页表填写的部分内容

    * 更新仓库网址

commit 3e3c6316aaac5a8a2932bd1746ec8b900dc5e2c6
Author: Chiichen <chiichen@qq.com>
Date:   Sun Jan 21 01:38:45 2024 +0800

    fix: 修复了ps2和tty初始化顺序的错误 (#497)

    * 修复了ps2和tty初始化顺序的错误

commit 74ffde667e5e7f4ac8ce6d5a5ec2c1403f36cbb0
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Jan 21 01:21:55 2024 +0800
```

(下页继续)

(续上页)

```
    feature: 增加early io remap的fixmap功能 (#495)

    允许在内存管理初始化之前,使用fixmap功能,映射一些物理内存,并记录.

commit 1f58c8f5cff8e82441b66789c3dc7c009d52f29a
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Jan 18 22:39:58 2024 +0800

    Update mini-backtrace版本到e0b1d90940 (#494)

    内容:

    [https://github.com/DragonOS-Community/mini-backtrace/pull/1](https://github.com/
↪DragonOS-Community/mini-backtrace/pull/1)

    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>

commit c75ef4e2126c180bf04c08635ffa5a278619c035
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Jan 18 00:09:36 2024 +0800

    添加early ioremap支持 (#492)

    * 使用early io remap来映射早期的vesa缓冲区

commit d8e29bffeee4fe4fe76ead3c761dd03f5395e6c2
Author: R0ronoa <84278015+2447742618@users.noreply.github.com>
Date:   Wed Jan 17 23:57:49 2024 +0800

    增加serio总线和相关trait (#488)

    * 新增serio总线和相关trait

    * 补充SerioDeviceManager和SerioDriverManager

commit 6994f6b113f6fea7b997ec07130a7bdaecfd67b7
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Jan 15 18:13:22 2024 +0800

    完成bitmap的static bitmap功能, 能够静态声明bitmap (#490)

    * 完成bitmap的static bitmap功能, 能够静态声明bitmap
```

(下页继续)

(续上页)

```
commit dcf232f378b36fad754799fc121a70cadc8d5cb3
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Jan 14 17:00:42 2024 +0800

    当找不到内核日志缓冲区的时候，重试 (#491)

commit 45626c859f95054b76d8b59afcbd24c6b235026f
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Jan 3 18:00:47 2024 +0800

    riscv: 解析dtb，获取可用内存空间并添加到memblock (#486)

commit 02343d0b5b47c07e7f4ec3818940795b1009fae1
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Jan 2 14:16:10 2024 +0800

    增加/dev/fb0，能够在用户程序读写帧缓冲区 (#485)

commit e7071df6a47c100381a8bc2000022e82d422361a
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Jan 1 11:53:49 2024 +0800

    把opengrok.ringotek.cn替换为code.dragonos.org.cn (#484)

commit c566df451ce6dbf2af684333e68b39fdfff86498
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Jan 1 11:46:51 2024 +0800

    添加帧缓冲区抽象并实现vesafb的驱动 (#483)

    - 添加bootparams对象
    - 修正由于bus的driver、device强弱引用关系 不正确从而导致对象被释放的bug
    - 添加vesafb的驱动
    - 实现framebuffer抽象层
    - 为通用帧缓冲区抽象实现sysfs的属性
    - 修改设备号DeviceNumber的定义
    - 仿照linux，添加initcall，并在第一个内核线程中，调用他们。

commit e3eb08d4d7148d6dad369e2ef27979d1bcf85bd6
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Dec 30 16:23:26 2023 +0800

    fix: 修复安装musl-gcc的脚本没能正确设置x86_64下的环境变量的问题 (#482)
```

(下页继续)

(续上页)

```
commit 81294aa2e6b257f0de5e3c28c3f3c89798330836
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Dec 27 20:32:25 2023 +0800

    fix: 修正bootstrap脚本安装docker后会使得当前终端进入root的问题 (#481)

commit cfd642e2835eded086b6e944427e4a88f03e2fff
Author: MemoryShore <105195940+MemoryShore@users.noreply.github.com>
Date:   Wed Dec 27 15:07:01 2023 +0800

    更新nova shell的revision为64ad1b282a (#477)

    - 修复tab补全时始终基于根目录的问题
    - 修复touch命令提示已存在文件的bug

commit 5e948c56506aa0554d212341a7630587d55ebb87
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Wed Dec 27 15:02:29 2023 +0800

    修正pipe逻辑，将pipe接入epoll。 (#478)

commit 0d6cf65aa124ee55bfee44cbb5196917ea6522fa
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Dec 27 14:27:12 2023 +0800

    Patch fix sched and net lockdep error (#479)

    - fix: 修复调度器,软中断,定时器,网络子系统的部分锁的使用不符合锁依赖安全规范的问题
    - fix: 修复创建pcb时内核栈爆栈的问题
    - 把异常的trap gate改成intr gate

    ---------

    Co-authored-by: GnoCiYeH <heyicong@dragonos.org>

commit 91e9d4ab55ef960f57a1b6287bc523ca4341f67a
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Dec 25 23:12:27 2023 +0800

    实现unified-init库，支持收集初始化函数到一个数组，并统一初始化 (#474)

    * 添加“统一初始化”的过程宏，并把SystemError独立成crate
```

(下页继续)

(续上页)

```
    * 使用unified-init来初始化fbmem

    * 更新workflow，增加内核自动化静态测试

commit f110d330d5493f383067b4e82ebbfb72f40457b2
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Dec 25 21:54:00 2023 +0800

    修复bootstrap在安装riscv gcc依赖时出现冲突的问题 (#476)

commit 406099704eb939ae23b18f0cfb3ed36c534c1c84
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Dec 25 18:08:12 2023 +0800

    增加epoll机制 (#455)

    * ## 增加epoll机制
    - 增加epoll机制
    - 添加事件等待队列，提升socket性能
    - 优化poll，删除不能poll的文件系统中的poll方法

    * 添加细节注释

    * 修复文件关闭后epoll还持有对应描述符的文件弱引用的bug

    * 将EPollEvent设计为POSIX标准

    * 修改s到us转换的计算错误

commit 070e991008b268b7103236a46b8651a983522869
Author: R0ronoa <84278015+2447742618@users.noreply.github.com>
Date:   Fri Dec 22 16:01:23 2023 +0800

    解决由于Makefile问题导致make run-uefi无法正常启动的问题 (#473)

commit 08a2ee408498b0db4c76c57b149f1cf047758f3c
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Dec 20 17:24:05 2023 +0800

    添加FrameBuffer的接口抽象&完善设备驱动模型的class相关代码 (#472)

    * 添加FrameBuffer的接口抽象（参考Linux 6.1.9）
```

(下页继续)

(续上页)

```
    * feature: 完善设备驱动模型的class的抽象,并创建graphics class

    * feature: 完善设备驱动模型中Device对class的处理,使得能够在class下注册设备

    目前注册了fbcon设备,但是由于虚拟终端还没写,
↪因此fbcon的到终端以及帧缓冲区的映射还没加上去.

commit 8612b6ce7afc903999ccf0b65bd65019484d2fad
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Dec 19 11:56:14 2023 +0800

    bugfix: 修复无法sleep的问题以及进程处于block(true)状态时无法被信号唤醒&
↪唤醒后不处理信号的问题 (#470)

commit 24ff1faffb3d610cd55e3c658fd50ea0a0efedfb
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Dec 18 17:44:53 2023 +0800

    doc: 修改Rust代码注释风格文档 (#471)

commit 111c5407ccb7774695c8047cf895481d3387fda9
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Dec 17 21:08:03 2023 +0800

    设置idle进程的时间片为0, 降低调度延迟 (#469)

commit 666cffedab3da9684e5abf5eebcbddae63590364
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Dec 16 22:26:26 2023 +0800

    riscv: 映射内核到指定的虚拟地址, 使得kinfo能正常工作 (#468)

    * riscv: 映射内核到指定的虚拟地址, 使得kinfo能正常工作

commit cf442324231632df3c3b0da3d2c8e19087863aa0
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Dec 13 14:44:57 2023 +0800

    修复x86下第二次编译的时候内核没有拷贝到磁盘的问题 (#467)

commit 1a72a751b18cf5bbe7b5b9e91aff530de0c18501
Author: LoGin <longjin@DragonOS.org>
```

(下页继续)

(续上页)

```
Date:   Thu Dec 7 02:13:22 2023 +0800

    在riscv输出hello world (#466)

    增加了以下内容:
    - SBI驱动
    - 把内核的rust工具链升级到2023-08-15版本
    - 输出riscv的helloworld
    - 设置内核是PIC的

commit fca83acef4ba261453f7c71960eacf83d0cf51f4
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Dec 4 22:10:10 2023 +0800

    修复cache-toolchain.yml的格式问题 (#464)

commit 6c7f966c2f3e3ed299168fc97ae346a85cc6e322
Author: MemoryShore <105195940+MemoryShore@users.noreply.github.com>
Date:   Mon Dec 4 22:07:30 2023 +0800

    NovaShell替换为默认shell (#456)

    * NovaShell替换为默认shell

    * delete some envvar

    * 自动从dragonos镜像站更新dadk

    * 更新github ci环境

    * 修复yml格式问题

    * 更新novashell到95738b235f

    ---------

    Co-authored-by: longjin <longjin@DragonOS.org>

commit 09d2bf52a6d048929a879748ae26e4fdea45e5d5
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Dec 3 21:27:43 2023 +0800

    update-dragon-stub-bf2617 (#463)
```

(下页继续)

(续上页)

```
commit af3543100543b9ac6159dc9f9367a76ff670b1f3
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Dec 3 17:16:03 2023 +0800

    使用submodule引入DragonStub (#462)

    * 修正构建系统文档：使用repo工具克隆代码

    * 使用submodule管理

commit 83ed0ebc293d5a10245089f627f52770fd5b9dd4
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Dec 3 14:51:21 2023 +0800

    修正构建系统文档：使用repo工具克隆代码 (#461)

commit 01090de77ef263b81edf449b77320d5fa28569de
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Dec 3 14:40:13 2023 +0800

    使用DragonStub引导riscv下的DragonOS内核 (#460)

commit 4fda81ce81939d83b74c8042d6fb4223deff3685
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Nov 25 12:07:39 2023 +0800

    使得DragonOS kernel 能为riscv64编译通过(尚未能启动) (#457)

    * 使得DragonOS kernel 能为riscv64编译通过(尚未能启动)

    * 修正了系统调用号声明不正确的问题，同时添加了编译配置文档

commit a1fd1cf1cbe6934221f95213ce02b21f21add225
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Nov 23 21:12:16 2023 +0800

    把tar的二进制镜像源更换为国内源 (#458)

commit cc5feaf67b914ecf701abcba70c01da149755491
Author: Jomo <2512364506@qq.com>
Date:   Thu Nov 23 21:04:32 2023 +0800
```

(下页继续)

(续上页)

```
    bugfix: 修复因rsdp v1 v2版本问题，导致ACPI无法正常初始化的bug (#454)

    bugfix: 修复因rsdp v1 v2版本问题，导致ACPI无法正常初始化的bug

commit c89d0c12377cd406a9b7465d7c087aeb9faefa51
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Nov 21 20:24:43 2023 +0800

    修复bootstrap的一系列脚本忘了source最新的shell rc的问题 (#453)

commit c75089286e9d49cef8d039446bf570c1bd4d2550
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Nov 21 13:42:18 2023 +0800

    调整脚本,使得能够创建riscv的磁盘镜像,并引导进入riscv下的grub (#450)

    * 安装musl toolchain以及riscv相关的工具链

    * 调整脚本,使得能够创建riscv的磁盘镜像,并引导进入riscv下的grub

    ```shell
    export ARCH=riscv64
    make write_diskimage
    make qemu
    ```

    即可在serial_opt.txt看到进入grub的提示信息

commit 48a3baa9b1d24dd1fc037cd9961945708c5c9b71
Author: LoGin <longjin@DragonOS.org>
Date: Tue Nov 21 13:42:06 2023 +0800

    安装musl toolchain以及riscv相关的工具链 (#449)

commit 84e7f7100664123d8ebc6b0f983c96242d15c396
Author: LoGin <longjin@DragonOS.org>
Date: Mon Nov 20 15:51:41 2023 +0800

    添加赞助商雅安数字经济运营有限公司的信息到readme (#451)

commit 99dbf38d2e279ea70e9e186753fd37001dbb749f
Author: LoGin <longjin@DragonOS.org>
Date: Sun Nov 19 11:42:53 2023 +0800
```

(下页继续)

(续上页)

```
    bugfix: 当物理机具有多个memory area的时候,无法正确使用这些区域的问题.
↪以及在内核代码处出现内存空洞而导致无法正常运行的问题. (#448)

    * bugfix: 当物理机具有多个memory area的时候,无法正确使用这些区域的问题.
↪以及在内核代码处出现内存空洞而导致无法正常运行的问题.

    解决方案:
    1. 分区域把空闲页添加到buddy
    2. 将内核链接到16M的位置,以避免uefi带来的内存空洞.

    这个值是因为我看到linux的救援内核也是在16M的地址,因此猜测厂商不会使用这块内存.
    尽管uefi规范讲的是固件可以采用任何地址,
↪内核需要使用内核重定位技术去避免遇到内存空洞,但我没有这么做.

commit 46e234aef65c081393fb7652e0ad2bae26786ce4
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Nov 17 21:25:15 2023 +0800

    使用cargo管理一些C文件的编译, 并且移动部分汇编到arch目录 (#447)

    * 使用cargo管理main.c的编译

    * 使用build-scripts编译架构相关的c代码

    * 删除elf.h

commit e4600f7f7d8f2295dbf970812ab1fcab81eb6eae
Author: Jomo <2512364506@qq.com>
Date:   Fri Nov 17 21:23:01 2023 +0800

    Kconfig (#432)

    * 内核编译配置

    * 将kernel.config的解析代码搬入crate

    * 将设置feature函数放入CargoHandler中

commit 11f78b73e7b18ef04e05e63612f8027eda0740e7
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Nov 17 20:05:57 2023 +0800
```

(下页继续)

(续上页)

```
    使用kernel-build脚本来编译所有的asm文件 (#445)

commit e4fb6c9754e535861a5e67db29fb6e8c2e9c8469
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Nov 17 12:26:10 2023 +0800

    美化readme (#446)

    * 美化readme

commit e26ca418df7af685226d12d7f22fe1785ba163e4
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Nov 17 11:26:26 2023 +0800

    把内核构建脚本单独独立成一个crate (#444)

commit a0c98cd4df88474f2efd927930862df50e016b73
Author: Jomo <xuzihao@dragonos.org>
Date:   Thu Nov 16 21:37:04 2023 +0800

    解决textui framework初始化成功后串口无法正常换行 (#443)

commit edaf015400f83967c2fc940f07be0dbb5792246f
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Nov 15 17:17:56 2023 +0800

    默认安装gnu tar到dragonos的/usr/bin目录下 (#442)

commit 0fb515b011967be01006cf88d788793dbbce2967
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Nov 15 15:39:35 2023 +0800

    完善pipe系统调用以及openat系统调用 (#441)

commit bf4a48994a2b284ee34aa49a66b4dec1b6ebc07c
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Nov 13 23:02:21 2023 +0800

    新增系统调用，并对照linux-6.1.9改写sys_wait4 (#440)

    * 1. 新增以下系统调用
                - SYS_LSTAT
                - SYS_READV
```

(下页继续)

(续上页)

```
                - SYS_ACCESS
                - SYS_UNLINK
                - SYS_CHMOD
                - SYS_FCHMOD
                - SYS_UMASK
                - SYS_SYSINFO
                - SYS_CLOCK_GETTIME
                - SYS_FCHMODAT
                - SYS_FACCESSAT

    2. 修改sys_wait4,使得其部分符合Linux的行为(还是有些地方不符合的,详情请对比linux-6.
↪1.9的sys_wait4接口)

commit 9b0abe6da72176086c3188e0599fda950562668f
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 21:23:48 2023 +0800

    添加access、faccessat、faccessat2 (#439)

commit 0d9b7d9240ef65c3e603a371db57a80d26a7b9dd
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 18:44:15 2023 +0800

    添加prlimit64系统调用 (#438)

    注意: 目前仅支持读取默认的rlimit值,尚不支持设置rlimit值.

commit 4a2d7191a3a6208b72e1d163c0235f566720f79a
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 17:53:36 2023 +0800

    bugfix: 解决shell在exec的时候传递的argv不正确的bug (#437)

commit 709498cac1f2134b2a5e089366ee7136ee029369
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 17:40:45 2023 +0800

    feat: sys_readlink && sys_readlinkat (#436)

commit be8cdf4b8edcd9579572672411f4489039dea313
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 16:36:17 2023 +0800
```

(下页继续)

(续上页)

```
    增加getrusage，并把apic timer的频率调整为系统HZ (#435)

commit 02e249f30bfe08b8a5cde1226ca0161f9e370927
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 14:11:33 2023 +0800

    添加uid、gid的系统调用（暴力封装返回0） (#434)

commit ea8ad4d42e52016fe581a2451165146f109dfd6e
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Nov 12 13:40:17 2023 +0800

    修正fork的时候没有正确拷贝vm holes的bug (#433)

commit c47fe90440fb7c6c82953e28a4b9597b22924758
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Nov 9 18:20:27 2023 +0800

    增加accept4系统调用 (#431)

commit 393f691574844544e76231379e4938e9046be7b9
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Nov 9 16:48:45 2023 +0800

    增加gettid以及线程组group leader相关的逻辑 (#430)

    * 增加gettid以及线程组group leader相关的逻辑

commit 0facf623d638816d7d01a700e19a52c3c16a8fa1
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Nov 9 00:10:34 2023 +0800

    修正文件open和写入的错误 (#429)

    1. 修正文件open的时候可能错误的把inode清空的问题(如果当前inode是mknod创建的)
    2. 修正fat和block device中,对文件写入部分的错误问题

commit 04babc3faba81997149ed11fda2ed03b4bbf4700
Author: MemoryShore <105195940+MemoryShore@users.noreply.github.com>
Date:   Wed Nov 8 21:42:51 2023 +0800

    实现fat文件系统的truncate方法 (#428)
```

(下页继续)

(续上页)

```
commit df2f5051ac645f600f2aefcaff3a9608b2c0de3f
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Nov 8 20:01:51 2023 +0800

    添加read the docs yml文件 (#427)

commit 5eaf536d5b81c234f9aea560e0c9d994fac3eb76
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Nov 8 19:41:08 2023 +0800

    添加初始化DragonOS的Rust-Musl工具链的脚本. (#426)

commit 7b32f5080f42bcbf7d2421013f3ea53c776a063c
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Nov 7 21:39:27 2023 +0800

    增加内存分配日志监视器 (#424)

    * 完成内存日志监视, 并输出日志到文件
    * 修复进程退出后,procfs查看进程status文件会崩溃的问题
    * 修复signal唤醒进程的判断条件问题

commit 70a4e5550a9fb49b537092287c3ddc36448c5b78
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Nov 7 20:32:06 2023 +0800

    使用rust重写了apic的驱动 (#425)

    * 使用rust重写了apic的驱动。

    * 修正signal和调度器的部分加锁逻辑, 增加回退策略。

    * 把pcb的flags字段替换为无锁的

    * 使用cargo管理apic的编译

    * 删除makefile中指定PIC的变量

    ---------

    Co-authored-by: Gou Ngai <ymd7823@outlook.com>
    Co-authored-by: 櫻井桃華 <89176634+TihayaKousaka@users.noreply.github.com>
```

(下页继续)

(续上页)

```
commit 4935c74f326cd4e0854959c0ec8ab1d726c05e41
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Nov 6 17:27:05 2023 +0800

    添加自定义的crc库(支持crc64) (#423)

commit 1effcfe519c06f04303340281fe9f62096184a74
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Sun Nov 5 23:15:46 2023 +0800

    修复readdir以及读磁盘时buf传错问题 (#422)

    * 修复readdir以及读磁盘时buf传错问题

    * fix potential memory problem

    ---------

    Co-authored-by: longjin <longjin@DragonOS.org>

commit 1603395155fc166de0ac5f80369526e196526ed2
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Sat Nov 4 21:39:44 2023 +0800

    支持syscall快速系统调用指令 (#417)

    * 支持syscall快速系统调用指令

    ---------

    Co-authored-by: LoGin <longjin@DragonOS.org>

commit 2f6f547ae05c19871138e558ba6943ff07f4c68c
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Sat Nov 4 21:35:25 2023 +0800

    Patch fix sched (#419)

    1.解决waitqueue sleep的时候,由于preempt count不为0,导致sched失败,
↪从而导致该waitqueue下一次wakeup时,会把pcb多次加入调度队列的bug
    2.修复socket inode 的read和write方法里面没有使用no_preempt的问题
    3. 修复cpu0的内核栈由于脏数据导致new_idle的时候set pcb报错的问题
```

(下页继续)

(续上页)

```
    ---------

    Co-authored-by: longjin <longjin@DragonOS.org>

commit 8058ccb307bbaf06a5810af32bcba3e41ab9fb93
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Nov 3 21:01:09 2023 +0800

    修复bootstrap.sh未能自动更换gcc镜像源，未能自动安装docker的问题 (#418)

commit d470019b1e675a04473cbb3c3eeaf180c8665e6d
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Nov 1 22:12:19 2023 +0800

    patch add mini backtrace (#416)

    * support rust panic backtrace

    mini-backtrace has llvm's unwind cpp source to support backtrace/unwind.
    as unwind/backtrace needs dynamically allocates memory, mini-backtrace
    uses stack memory to capture fixed number of backtrace to avoid heap
    allocation.
    as unwind library needed, it needs to turn on eh_frame_hdr

    * 修改忘了生成kernel.elf的问题

    * 设置backtrace是默认的feature

    ---------

    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>

commit 8b3d1688daac2aaf4e87403ecda5467a01464f81
Author: yuyi2439 <68320855+yuyi2439@users.noreply.github.com>
Date:   Wed Nov 1 21:11:55 2023 +0800

    把pci驱动的读取acpi mcfg的代码，调整为从新的acpi驱动来读取 (#413)

    * 把pci驱动的读取acpi mcfg的代码，调整为从新的acpi驱动来读取

commit 971462be94ba0a5c74af7a5f9653dfabd4932a63
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Wed Nov 1 20:55:57 2023 +0800
```

(下页继续)

(续上页)

```
    添加thread和futex机制 (#411)

    * 初步实现clone系统调用

    * 实现了线程，初步实现futex机制，添加了几个小的系统调用

    * 更改pcb引用计数问题

    * 解决死锁bug

    ---------

    Co-authored-by: LoGin <longjin@DragonOS.org>

commit 665f4a7707e33f3b4d2fde77113fa3d13b5b52c4
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Wed Nov 1 14:18:00 2023 +0800

    更新了使用clangd的.gitignore (#415)

    Co-authored-by: chiichen <chiichen@qq.com>

commit 77799ccaaca276fe127448d169f0e035837cce44
Author: Wu Mianzhi <31810920+Hdksg10@users.noreply.github.com>
Date:   Mon Oct 30 00:08:52 2023 +0800

    完成e1000e驱动 (#393)

    * 测试RESET

    * 测试RESET

    * 基于轮询的实现

    * 规范化部分unsafe的使用

    * 完成中断处理函数，同时去除了不必要的内存拷贝行为，准备编写napi机制

    * 实现现有协议栈下的部分napi机制；修复了内存泄漏的问题；添加了一部分代码注释

    * 去除部分无用代码
```

(下页继续)

(续上页)

```
    * 去除一些无用代码

    * 适配新的驱动模型

    * 完成msi中断测试

    * 去除一些无用代码

    * 格式化代码

    * 增加了一些注释，提高代码可读性

    * 去除无关文件

    * 优化了读取mac地址的方式，提高可读性

commit fbe6becd6dd3cd72643707e0088f20364ac1b166
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Oct 26 23:08:39 2023 +0800

    添加rust重构版本的HPET驱动和tsc驱动,并使用HPET校准tsc频率和cpu总线频率 (#412)

    * 添加rust重构版本的HPET驱动和tsc驱动,并使用HPET校准tsc频率和cpu总线频率

    * 把hpet.c移动到arch文件夹下

commit ad1d649eddee4aa8ac81b2f44bc99da462a6a813
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Tue Oct 24 19:59:01 2023 +0800

    更新系统调用号 (#410)

    * 更新系统调用号

    * 更改DragonReach和relibc版本

    * update

    * update

    * fix warning

    ---------
```

(下页继续)

(续上页)

```
    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit f4082b86b15989a1d43e62050c6ba9b363c91ece
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Oct 24 16:40:49 2023 +0800

    更改系统调用的寄存器传参顺序 (#409)

commit 40314b30ab2a7e1fd06a05a00f693e644e446035
Author: Xiaoye Zheng <xiaoyez@zju.edu.cn>
Date:   Tue Oct 24 14:31:56 2023 +0800

    DragonOS虚拟化 (#389)

    * try some ioctl flow & kvm device

    * add sys ioctl

    * 删掉一些debug信息

    * 修改run-qemu.sh脚本，在QEMU中enable vmx

    * 修改cr0,cr4,msr寄存器enable VMX operations

    * enable vmx operation

    * allocate memory for vmcs with bug

    * allocate memory for vmcs

    * cpu virt-50%

    * single vcpu virt

    * add vmcs fields

    * CPU virt overall flow with bug

    * run vmlaunch success

    * run CPU virt with bug
```

(下页继续)

(续上页)

```
    * 成功运行non-root模式的guest

    * 成功运行vmexit,进入vmx_return函数

    * 成功运行vmlaunch, vmexit, vmresume

    * vmexit handler with bug

    * 完成vmexit cpuid handler

    * fix vmresume guest状态恢复的bug

    * 增加vm ioctl

    * refactor kvm 50%

    * refactor kvm 80%

    * FIXME: kvm vmlaunch failed

    * vmlaunch success

    * FIXME: output error

    * update guest_rsp

    * cpu virt refactor

    * add mmu related struct

    * add usermemory region workflow

    * add mem-virt workflow

    * add mem-virt

    * refactor code

    * add vcpu ioctl set_regs

    * rename hypervisor to vm & solve some deadlock bugs

    * workout mem pipeline
```

(下页继续)

(续上页)

```
    * fix vmcs control setting bugs

    * refactor segment regs initialization

    * resovle conficts

    * resovle conficts

    * format code

commit 485e2487616b1d33776b63724d4abc1ae8f506e8
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Oct 24 14:19:26 2023 +0800

    修改脚本,只有当磁盘未安装Grub的时候,才执行grub-install. 节省编译时间 (#408)

commit 46795849a29eef77fd6f7af548d05ee6e654c5bb
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Oct 24 13:56:57 2023 +0800

    修复bootstrap.sh安装顺序导致的问题 (#407)

commit 3c82aa56d1b784ea7371100b3e906365be8332fd
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Tue Oct 24 12:02:20 2023 +0800

    Signal refactor (#402)

    * 初步完成对 signal_types 和 部分signal代码的初始化

    * 重构了一部分架构相关代码进入 arch 中

    * 基本修改完成，编译通过，后续补上系统调用

    * signal基本完成，能实现 Sigaction 系统调用

    * 增加了一组枚举抽象

    * 进一步重构了一部分C风格的代码

    * 继续重构了一部分C风格代码
```

(下页继续)

(续上页)

```
* 继续完善了一部分逻辑

* 修改了部分代码逻辑

* 补充了 fork 中复制信号信息的逻辑

* 修复了 kallsysms 未转义引号的问题

* 修复了无法跳转到 sigreturn 的bug

* 调通了 signal

* 实现了 signal 架构抽象层的 trait

* 为信号提供了默认处理函数

* 基本完成了 signal 的大体逻辑

* 修复了 Sigreturn 的一个小错误，格式化

* 修复了一个编译器漏报错误

* 删除了多余的代码

* 修改测试程序为链接 relibc

* 修复了信号处理过程中浮点寄存器错误保存的问题

* 修复了一个结构体错误引起的无法在relibc下正确运行的错误

* 修复了链接 relibc 时无法正常从信号处理返回的 bug

* 修复了 signal 处理流程中 rsp 指针错误导致的浮点运算触发GP

* 修复了一个死锁问题，解决了默认处理函数无法进入调度导致的bug

* 修复了一些错误

* 修改了 relibc 依赖版本号

* 删除了多余的 imports

* 删除一些debug日志
```

(下页继续)

(续上页)

```
    * 删除内核 signal.h 文件

    * 删除一个依赖项

    * 删除了 binding 相关依赖项

commit d7f5742a206c6c25ed30009796eb8248429f0a1e
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Oct 23 21:40:39 2023 +0800

    初步编写cpu信息获取的代码 (#406)

    1. 启动时从acpi获取所有的cpu信息并存到SMP_BOOT_DATA
    2. 注册cpu subsystem/bus到sysfs(暂时未添加内容)

    todo:
    1. build_cpu_map(在X86_64SmpManager中)
    2. 实现cpu mask
    3. 把cpu设备注册到sysfs

commit 7eda31b2f07c6ef41dc0d2bd13051f0fce5e5976
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Oct 22 22:00:16 2023 +0800

    在Sysfs中引入ACPI Firmware (#405)

    - bugfix: multiboot2启动的信息因为没及时转存导致后面无法从其中进行查询的bug
    - feature: 把acpi表、acpi bus加入sysfs

commit 01bd5258cf467326819c77584713fbc6ffe4fb32
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Oct 22 12:22:41 2023 +0800

    解决shell无法输入大写字母'P'的问题 (#404)

commit a03c4f9dee5705207325c56629c0ccd219168f10
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Oct 20 22:11:33 2023 +0800

    设备驱动模型：完善platform bus相关内容。并注册串口到sysfs (#403)

    * 完成初始化platform bus
```

(下页继续)

(续上页)

```
    * 删除旧的sysfs
    * 把uart驱动移动到tty/serial文件夹下
    * 完成将串口挂载到sysfs
    * 修复vfs系统调用未能follow symlink的问题
    * 修复shell未能正确获取pwd的问题

commit 06d5e247267cb65b84a80f219853ccd0f384b16e
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Oct 11 00:53:15 2023 +0800

    完善设备驱动模型，基于kset、kobj来维护对象之间的关系 (#401)

    * 使用kobj和kset管理/sys文件夹下的对象

    * 修改notifier，把action从u64换为泛型。

    * 完善设备驱动模型，基于kset、kobj来维护对象之间的关系

commit 6abb8bd7c0ee7746f0b6cf682a0c4d112a2ef6a3
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Oct 9 01:55:58 2023 +0800

    在github workflow的工具链配置文件里面添加rust-src组件 (#400)

commit 9e9ffedfc59cd89f4cdbd7984abbb2ac19ea9575
Author: Plucky923 <107762234+Plucky923@users.noreply.github.com>
Date:   Mon Oct 9 01:11:14 2023 +0800

    syscall: 完善syscall代码 (#387)

    * syscall: 完善syscall代码

    修改代码使这段代码可以使用语法糖。修改SYS_READ和SYS_WRITE的安全检查为userbuffer

    Signed-off-by: plucky <m202372036@hust.edu.cn>

    * syscall: 修改SYS_READ和SYS_WRITE的权限检查为userbuffer

    Signed-off-by: plucky <m202372036@hust.edu.cn>

    * syscall: 有不知道如何修改的错误

    Signed-off-by: plucky <m202372036@hust.edu.cn>
```

(下页继续)

(续上页)

```
    * syscall: 修改SYS_READ和SYS_WRITE并编译通过

    Signed-off-by: plucky <m202372036@hust.edu.cn>

commit 865f4ba4cdce23b154844d6d297f75033f3dcf70
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Oct 9 01:10:14 2023 +0800

    修改shell执行exec时传参错误问题 (#399)

    * 修改shell执行exec时传参错误问题

commit b7b843beddea12cdedda90f6129b7c9980876112
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Oct 9 00:58:08 2023 +0800

    wait4系统调用支持options字段 (#398)

commit 2dbef7859f0af395ccec348f17cf0b79ed56e003
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Mon Oct 9 00:28:08 2023 +0800

    命名管道系统调用以及文件系统兼容特殊文件类型的接口 (#397)

    * 修复pipe2在读端或写端关闭后还阻塞问题。

    * 实现命名管道机制，增加特殊文件类型兼容普通文件系统的接口。

    * 普通文件系统能够适配特殊文件(命名管道等)

commit 34e6d6c80f36494088db3284f85d1a2c63aa18a8
Author: yuyi2439 <68320855+yuyi2439@users.noreply.github.com>
Date:   Sun Oct 8 14:26:17 2023 +0800

    实现free指令+修复 mountfs的内存泄露问题(#394)

    * 实现meminfo文件

    * 成功实现free指令，添加了一些string有关函数，并进行一些无影响的小改动


    * 解决内存泄露的问题： mountfs inode的wrap方法使用了Arc::into_raw而没有from_
↪raw，导致inode始终无法释放
```

(下页继续)

(续上页)

```
    ---------

    Co-authored-by: LoGin <longjin@DragonOS.org>
    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit afc95d5c2541c27c762091ad38fdffabe355db5a
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Tue Oct 3 12:09:29 2023 +0800

    完善pci中断的设计 (#392)

    * 完善pci中断的设计

commit 876cb89ecf7c1bf1646bfc392efcbafacad2262f
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Tue Oct 3 12:03:34 2023 +0800

    修复pipe2在读端或写端关闭后还阻塞问题 (#396)

    * 修复pipe2在读端或写端关闭后还阻塞问题。

    * update

    * update

    * 修改cloexec

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit fba5623183378da6b120caafca120615328efa2e
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Oct 2 20:46:19 2023 +0800

    引入intertrait库，支持trait之间的互相转换 (#395)

    * 能过编译（test还没法跑）

    * 初始化intertrait转换库

    * update license of intertrait
```

(下页继续)

(续上页)

```
commit bb0e4d4131046a69bbccdc0cdf1d5db51a2c6126
Author: GnoCiYeH <heyicong@dragonos.org>
Date:   Sat Sep 30 16:36:06 2023 +0800

    使用DragonReach启动shell，修改getdents (#391)

    * 使用DragonReach启动shell，修改getdents

    * 更改关闭pipe时断言报错问题，以及DragonReach启动shell阶段版本

    * 修改目录结构

    * update

    * 解决小问题

    * 调整dragon reach版本号

    * 设置make clean的时候不清空应用程序的缓存。
    指定relibc版本号

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 0dd8ff43325b494ea777dbe6e552fdc77b9dabc8
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Thu Sep 21 23:23:57 2023 +0800

    添加中断 (#370)

    * 添加中断

    * dhcp更改为全局socketset

    * 解决异常中断的问题，使得能够使用中断来处理网卡数据

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 6b4e7a2972cc06663754c0e35a0e541987006fa4
```

(下页继续)

(续上页)

```
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Sep 19 19:46:59 2023 +0800

    增加kernfs (#386)

    * 增加kernfs

    * kernfs文档

commit ae5ede03bebe5c4b593ad7a350f0945f1367be7c
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Sep 18 07:38:04 2023 +0800

    bugfix: bus/device manager对卸载逻辑的处理错误 (#385)

    * 移动位置

    * bugfix: bus/device manager对卸载逻辑的处理错误

commit 7ae679ddd6481897a86523a52fad3b060254fa5b
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Sep 17 15:41:01 2023 +0800

    ahci内存越界问题修复+ mm的bug修复+在rust中解析acpi table (#384)

    * bugfix: 修复了Flusher Drop的时候没有自动刷新TLB的bug

    * 解决进程管理未初始化时，trap.c尝试打印pid导致错误的问题

    * 设置kmalloc默认强制清0

    * 修复ahci驱动的内存越界问题
    * 修复mmio buddy忘记归还buddy block的问题
    * 新增acpi模块,暂时能解析acpi tables

commit 11110997465e858757da54b5ce28d7c22690aaff
Author: hanjiezhou <zhouhanjie@dragonos.org>
Date:   Sat Sep 16 20:14:56 2023 +0800

    修改 tty 中resize bug (#383)

commit 71474bc6829b3cd831df7ce24ea059557996524d
Author: LoGin <longjin@DragonOS.org>
```

(下页继续)

(续上页)

```
Date:   Sat Sep 16 16:16:43 2023 +0800

    修复drop fd时，文件描述符引用不为0的问题 (#382)

commit de71ec259cd21c782f4031b01635eb8ad3df1943
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Sep 15 19:44:11 2023 +0800

    修正由于init proc union导致的无法运行的问题 &&␣
↪修正由于内核线程启动后默认sleep的行为导致init进程无法正常运行的bug (#381)

    1. 修正由于init proc union导致的无法运行的问题
    2. 修正由于内核线程启动后默认sleep的行为导致init进程无法正常运行的bug

commit 1496ba7b24a5e6954291ca9643b9f3cec567479a
Author: LoGin <longjin@DragonOS.org>
Date:   Fri Sep 15 14:58:19 2023 +0800

    进程管理模块重构完成 (#380)

    * 添加新版pcb的数据结构 (#273)

    * 将pcb中的内容分类，分别加锁 (#305)

    * 进程管理重构：完成fork的主体逻辑 (#309)

    1.完成fork的主体逻辑
    2.将文件系统接到新的pcb上
    3.经过思考,暂时弃用signal机制,待进程管理重构完成后,重写signal机制.
↪原因是原本的signal机制太烂了

    * chdir getcwd pid pgid ppid (#310)


    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

    * 删除旧的fork以及signal的代码，并调整fork/vfork/execve系统调用 (#325)

    1.删除旧的fork
    2.删除signal相关代码,等进程管理重构结束之后,再重新写.
    3.调整了fork/vfork/execve系统调用
```

(下页继续)

(续上页)

```
    * 实现切换进程的代码 (#331)



    * 实现切换进程的代码

    * Patch modify preempt (#332)

    * 修改设置preempt的代码

    * 删除rust的list和refcount

    * 为每个核心初始化idle进程 (#333)

    * 为每个核心初始化idle进程

    * 完成了新的内核线程机制 (#335)

    * 调度器的pcb替换为新的Arc<ProcessControlBlock>, 把调度器队列锁从 RwSpinLock␣
↪替换为了 SpinLock (#336)

    * 把调度器的pcb替换为新的Arc<ProcessControlBlock>

    * 把调度器队列锁从 RwSpinLock 替换为了 SpinLock , 修改了签名以通过编译

    * 修正一些双重加锁、细节问题

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

    * github workflow自动检查代码是否格式化

    * cache toolchain yml

    * 调整rust版本的waitqueue中的pcb为新版的pcb (#343)

    * 解决设置rust workspace带来的“工具链不一致”的问题 (#344)


    * 解决设置rust workspace带来的“工具链不一致”的问题
```

(下页继续)

(续上页)

```
    更改workflow

    * 调整pcb的sched_info和rwlock，以避免调度器死锁问题 (#341)

    * 调整pcb的sched_info和rwlock，以避免调度器死锁问题

    * 修改为在 WriterGuard 中维护 Irq_guard

    * 修正了 write_irqsave方法

    * 优化了代码

    * 把 set state 操作从 wakup 移动到 sched_enqueue 中

    * 修正为在 wakeup 中设置 running ，以保留 set_state 的私有性

    * 移除了 process_wakeup

    * 实现进程退出的逻辑 (#340)

    实现进程退出的逻辑

    * 标志进程sleep

    * 修复wakeup的问题

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

    * rust 重构 completion (#350)

    * 完成了completion的基本结构，待完善上级调用

    * 用SpinLock保护结构体并发安全

    * 修改原子变量为u32，修复符号错误

    * irq guard

    * 修改为具有内部可变性的结构体

    * temp fix
```

(下页继续)

(续上页)

```
* 修复了由于进程持有自旋锁导致的不被调度的问题

* 对 complete 系列方法上锁，保护 done 数据并发安全

* 移除了未使用的依赖

* 重写显示刷新驱动 (#363)

* 重构显示刷新驱动

* Patch refactor process management (#366)

* 维护进程树

* 维护进程树

* 更改代码结构

* 新建进程时，设置cwd

* 调整adopt childern函数，降低开销

---------

Co-authored-by: longjin <longjin@RinGoTek.cn>

* waitqueue兼容C部分 (#351)

* PATH

* safe init

* waitqueue兼容C部分

* waitqueue兼容C部分

* 删除semaphore.c,在ps2_keyboard中使用waitqueue

* 删除semaphore.c,在ps2_keyboard中使用waitqueue

* current_pcb的C兼容
```

(下页继续)

(续上页)

```
    * current_pcb的C兼容

    * current_pcb的C兼容

    * fmt

    * current_pcb的兼容

    * 针对修改

    * 调整代码

    * fmt

    * 删除pcb的set flags

    * 更改函数名

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

    * merge master

    * Patch debug process management refactor (#372)

    * 能够调通，执行完textui_init

    * 能跑到initial kernel thread

    * fmt

    * 能够正常初始化所有服务（尚未能切换到用户程序）

    * 删除部分无用的extern

    * 存在问题：ap处理器启动后，bsp的smp_init函数return之后就出错了，怀疑是栈损坏

    * 解决smp启动由于未换栈导致的内存访问错误

    * debug

    * 1
```

(下页继续)

(续上页)

```
* 1

* lock no preempt

* 调通

* 优化代码，删除一些调试日志

* fix

* 使用rust重写wait4 (#377)

* 维护进程树

* 维护进程树

* 更改代码结构

* 新建进程时，设置cwd

* 调整adopt childern函数，降低开销

* wait4

* 删除c_sys_wait4

* 使用userbuffer保护裸指针

---------

Co-authored-by: longjin <longjin@RinGoTek.cn>

* 消除warning

* 1. 修正未设置cpu executing的问题

* 修正kthread机制可能存在的内存泄露问题

* 删除pcb文档

* 删除C的tss struct
```

(下页继续)

(续上页)

```
    ---------

    Co-authored-by: Bullet <93781792+GP-Bullet@users.noreply.github.com>
    Co-authored-by: Chiichen <39649411+Chiichen@users.noreply.github.com>
    Co-authored-by: hanjiezhou <zhouhanjie@dragonos.org>
    Co-authored-by: GnoCiYeH <118462160+GnoCiYeH@users.noreply.github.com>
    Co-authored-by: houmkh <1119644616@qq.com>

commit b087521e07f601b30e3d48df788fcc2f09f19566
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Wed Sep 13 18:01:52 2023 +0800

    完善设备驱动模型&调试串口驱动 (#379)

    * 完成了基本架构重构，正在进行兼容

    * 重构了所有 Device Driver ，还没有接上具体设备

    * 基本把 Uart 接上了，还没有测试

    * 初步完成系统设备初始化

    * 初步重构 BlockDevice ，使其兼容新的 Device 结构

    * 修改文件系统内的部分函数调用以满足重构后的接口

    * 测试完 Uart 设备的功能

    * 移除了自动添加的文件

    * 修复了 warning 和部分格式

    * 解决warning，并且修正sysfs初始化的位置

    * Patch fix

    * 删除了 sysinfo 的默认实现

    * 删除了字符设备读写的 offset 参数

    * 修复了 warning 和一些小逻辑错误

    ---------
```

(下页继续)

(续上页)

```
    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 9029414af2089cbe7d2d2097be2e116c09beb6dd
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Wed Sep 13 01:49:03 2023 -0400

    use demangled names (#375)

    there is no change for c symbols
      rust symbols name will be more readable

commit 22c9db312a5f02b48a1bf7853dc53434da65e28a
Author: hanjiezhou <zhouhanjie@dragonos.org>
Date:   Wed Sep 13 00:58:01 2023 +0800

    Patch pipe2 (#364)

commit 68312d3c68b9df288589f9636417745d46520ad2
Author: Xshine <gshine@m.scnu.edu.cn>
Date:   Wed Sep 13 00:26:41 2023 +0800

    修正造成 http server 的错误 (#378)

    * 修正造成 http server 的错误

commit 285de542f3ddb09bfdcad0a7022bd722f4f1cace
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Wed Sep 6 06:08:52 2023 -0400

    modify kernel link script (#373)

    put rust text between _text and _etext, so rust symbols are included in
    kallsyms, traceback could use them.

    modify grub_auto_install.sh to add arch linux support

commit 3b0bf43bbb196a74a64cee8ecd2d4eb884c5e9ee
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Mon Sep 4 00:57:52 2023 -0400

    fix compiler warnings in pci_irq.c (#371)
```

(下页继续)

(续上页)

```
    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>

commit 607783d7414735d1dc54afb0c7346ed8e13303a2
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Sun Sep 3 01:36:02 2023 -0400

    fix pci_irq.c - memory leak - wrong irq_name allocated length (#367)

    add function declaration to avoid compiling warning

    add extra packages need for build

    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>

commit d9113303d8e1d449a122f7a5f66453fbe7c26a46
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Sep 3 13:33:27 2023 +0800

    relibc与旧的C库同时开始编译 (#369)

commit 4895ff6968ae8f24c7a0d55dce6ae23082d60e3b
Author: yuyi2439 <68320855+yuyi2439@users.noreply.github.com>
Date:   Sat Sep 2 00:27:41 2023 +0800

    fix: DragonOS-Community/DragonOS#358 (#361)

commit 8479f19979aa9ddc4e383651cc56a7a24bd94e21
Author: GnoCiYeH <118462160+GnoCiYeH@users.noreply.github.com>
Date:   Fri Sep 1 21:46:36 2023 +0800

    添加rust-gdb调试内核文档 (#357)

    * 编写使用GDB调试内核文档

commit 863a3cff06e618a5f0fc03920dfd5732452344c9
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Aug 31 20:25:00 2023 +0800

    添加与rust std接口相同的once库 (#353)

commit a3ef8f8ad5248e3424113871950eb9c80eeeb99e
Author: GnoCiYeH <118462160+GnoCiYeH@users.noreply.github.com>
Date:   Thu Aug 31 19:52:32 2023 +0800
```

(下页继续)

(续上页)

```
    修改RamFS目前存在的BUG (#354)

    * 修改RamFS目前存在的BUG

commit c757940bd61b0125e037a59eb77565e42470201b
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Thu Aug 31 17:54:49 2023 +0800

    优化makefile (#352)

commit 2dd9f0c7503d1a325713764fedbce06fcab3a06b
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Aug 28 15:54:52 2023 +0800

    mmio␣
↪buddy新增guard，把映射的职责交由其守卫进行处理，并且守卫被drop的时候自动释放内存 (
↪#346)

    * mmio␣
↪buddy新增guard，把映射的职责交由其守卫进行处理，并且守卫被drop的时候自动释放内存

commit 8d94ea66a3eb3e02039730c8d08e9bead8c344b8
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Mon Aug 28 15:43:07 2023 +0800

    Patch ahci (#348)

    * Modify the ahci module and delete the useless c code

    修改ahci使其不再依赖旧的pci函数
    删除旧的pci、msi函数代码

commit f5df0e79c67a9508bc6a50fccded9dec78e7ed9d
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Aug 28 15:29:00 2023 +0800

    解决userbufferwriter的长度错误问题，并修复gettimeofday的pagefault问题 (#349)

    * 解决userbufferwriter的长度错误问题，并修复gettimeofday的pagefault问题

commit ddb9d91712b6e87aa15b9cc4a8fdec8ae0996a5e
Author: Xshine <gshine@m.scnu.edu.cn>
```

(下页继续)

(续上页)

```
Date:   Sun Aug 27 15:54:19 2023 +0800

    将 io 移动至 vfs 目录，并修正引用路径 (#339)

    * 将 io 移动至 vfs 目录，并修正引用路径

    * fix bug in makefile

commit e92d02281005bac31fe80e9070ac4a2a6cef0419
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Aug 26 21:36:13 2023 +0800

    解决设置rust workspace带来的“工具链不一致”的问题 (#345)

    更改workflow

commit 9a367aa7eb1576a235f5f52ee542132a1e5e39df
Author: LoGin <longjin@DragonOS.org>
Date:   Thu Aug 24 18:50:52 2023 +0800

    添加github workflow，检查代码是否已经格式化 (#342)

    * 添加github workflow，检查代码是否已经格式化

commit f09a98329c4ec77010de86d126516310b407455a
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Aug 23 16:09:29 2023 +0800

    1. 修复bootstrap.sh在安装libssl-dev之前，安装dadk，从而导致错误的问题 (#338)

    1. 修复bootstrap.sh在安装libssl-dev之前，安装dadk，从而导致错误的问题
    2. 构建系统的文档，补充对vnc端口的说明

commit 4537ffb7e9afb2d96f2adcee32c8ac84b056d2e5
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Mon Aug 21 18:37:31 2023 +0800

    实现了对用户空间传入指针抽象的UserBufferReader/
↪Writer，来检验用户空间指针地址并提供一定的功能抽象 (#326)

    * 构建了 Userbuffer 对用户空间传入的指针进行了抽象，并提供了读写操作

    * 分成了Reader和Writer，增加了从地址读和写入到指定地址的功能
```

(下页继续)

(续上页)

```
    * 删除了多余的注释

    * 增加了直接获取BufferWriter切片的函数

    * 通过 where 的一个 Trick 实现了 const generic 和后续功能

    * 替换为了 core::slice::align_to 实现&[u8]转&[T]

    * 移除了 userbuffer.rs

    * 提供了独立获取缓冲区中不同偏移量位置的数据的函数

    * 替换了部分系统调用(还未测试

    * 简化了代码

    * 修复内存越界的bug

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit abe3a6ea3c543425e2cad12722e8a658b324d515
Author: hanjiezhou <zhouhanjie@dragonos.org>
Date:   Sun Aug 20 00:19:36 2023 +0800

    Patch refactor scm and textui (#289)

    * 重构屏幕管理器和textui框架

    * 切换字体为spleen, 并增加对字体的抽象

    * 修正文档

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>
```

21.2 V0.1.8

备注： 本文作者：龙进 longjin@DragonOS.org

2023 年 8 月 16 日

21.2.1 贡献者名单

DragonOS V0.1.8 版本由以下小伙伴贡献代码：

- 龙进 longjin@DragonOS.org
- 侯嘉滢 houjiaying@DragonOS.org
- 吴宇健 wuyujian@DragonOS.org
- 黄厅 huangting@DragonOS.org
- 孔维超 kongweichao@DragonOS.org
- 蔡嘉鑫 caijiaxin@DragonOS.org
- 池克俭 chikejhian@DragonOS.org
- zhaoyao73 dragonlinux@gmail.com
- 周瀚杰 zhouhanjie@DragonOS.org
- Bullet 93781792+GP-Bullet@users.noreply.github.com
- 櫻井桃華 89176634+TihayaKousaka@users.noreply.github.com
- Tptogiar 2528891112@qq.com

21.2.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- 万晓兰
- David Wen
- Seele.Clover
- FindWangHao
- ferchiel
- 叶锦毅
- 林

- Albert
- TerryLeeSCUT · GitHub
- slientbard
- 悟
- 匿名热心人士

21.2.3 更新内容-内核

新特性

- refactor: 重构系统调用模块 (#267)
- feature: 添加 AlignBox 和 int_like 宏 (#272)
- refactor: 新的 ipi 功能 &kick_cpu 功能的重写 (#274)
- feature: 实现 gettimeofday() 系统调用和 clocksource+timekeeping 子模块 (#278)
- refactor: PCI 设备中断重构，并删去 USB 相关代码 (#285)
- feature: 注册串口设备，创建字符设备框架 (#290)
- refactor: 新的内存管理模块 (#303)
- feature: 新的二进制加载器、elf 解析器 (#303)
- feature: 增加 ListenTable 来检测端口占用 (#291)
- feature: 替换 local_irq_save 为 IrqFlagsGuard 实现 (#317)
- feature: 实现系统调用 Fstat (#295)
- feature: 实现内核通知链 notifier chain (#316)
- feature: 增加 fcntl 系统调用 (#323)
- feature: 添加 per cpu 变量支持 (#327)
- feature: spinlock 守卫新增 leak,spinlock 新增 force unlock 功能.(#329)

bugfix

- bugfix: 修复无法正常读取 stdin 的问题 (#264)
- bugfix: 修复了当传入 ahci 驱动的缓冲区地址为用户缓冲区时, 产生的内存越界问题.(采用分配内核缓冲区的方式临时解决) (#265)
- bugfix: 解决由于 local_irq_save、local_irq_restore 函数的汇编不规范导致影响栈行为的 bug。(#303)
- bugfix: 解决 local_irq_save 未关中断的错误 (#303)

- bugfix: 解决 arch_try_cmpxchg 对于指针处理的错误 (#307)
- bugfix: 修复了 wait4 的异常报错 (#312)
- bugfix: 修正 null 设备以及 zero 设备无法 open、行为不符合预期的问题 (#314)
- bugfix: 修正 fat 文件系统未能正确的扩展文件大小的 bug (#323)
- bugfix: 修正 rwlock 有的地方由于未使用 ManuallyDrop 导致的 use after free 问题 (#329)

21.2.4 更新内容-用户环境

新特性

- feature: 新增 http server (#265)

bugfix

- bugfix: 解决链接时，由于 crt*.o 未按照升序排列导致 init 段链接错误的问题 (#265)

21.2.5 更新内容-其他

- bugfix: 固定编译工具链、修复由于新版 rust 编译器问题导致的报错。(#258)
- feature: Makefile: 根目录下添加 make help 命令 (#271)
- doc: 更新 github issue 模板 (#277)
- bugfix: 解决 relibc 的头文件没能识别 __dragonos__ 定义的问题 (#315)
- feature: 设置内核、relibc 的远程为 dragonos 的 git 镜像站，防止国内网络问题导致编译失败 (#318)
- feature: 自动安装、更新 dadk (#319)

21.2.6 更新内容-软件移植

- feature: 移植了 sqlite3 (#323)

21.2.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/DragonOS-Community/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://git.mirrors.DragonOS.org
- 国内镜像加速：https://mirrors.ringotek.cn/

21.2.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT
- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT
- redox - https://gitlab.redox-os.org/redox-os/redox - MIT

21.2.9 当前版本的所有提交记录

```
commit 40176b1c6603d487b7eb66fb81e641f0932ab90a
Author: longjin <longjin@RinGoTek.cn>
Date:   Tue Aug 15 15:06:57 2023 +0000

    删除无用代码,并把about app的版本号更新为0.1.8

commit 67b481888770c6469f572f244a4f97e42da77d1f
Author: houmkh <1119644616@qq.com>
Date:   Mon Aug 14 12:18:46 2023 +0800

    移动fstat (#330)

    * 移动fstat

commit 90a0a49048fdaf5e31320d0c87f8bed8db1bd882
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Aug 14 01:24:49 2023 +0800
```

(下页继续)

(续上页)

```
    修正rwlock有的地方由于未使用ManuallyDrop导致的use after free &&␣
↪spinlock守卫新增leak,spinlock新增force unlock功能.(#329)

    1.修正rwlock有的地方由于未使用ManuallyDrop导致的use after free
    2. spinlock守卫新增leak,spinlock新增force unlock功能.

commit c3dad0011d331d782670e14723aa48e98fbac787
Author: LoGin <longjin@DragonOS.org>
Date:   Sun Aug 13 16:28:24 2023 +0800

    添加per cpu变量支持 (#327)

commit 42c97fa7f4fee7eeefeda5d2b7ed14f598a58493
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Aug 8 23:45:04 2023 +0800

    删除旧的libELF (#324)

commit 6d81180b3b7328466b976b69c5f7782aa66d8a89
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Aug 8 23:39:22 2023 +0800

    移植sqlite3,并修复一些bug (#323)

    * bugfix: 程序加载器映射内存时，计算要映射的大小不正确的问题。

    * 修正brk系统调用不符合规范的地方

    * bugfix: 修正fat文件系统未能正确的扩展文件大小的bug

    * 增加fcntl系统调用

    * 移植sqlite3

commit 26887c6334cdca2d13ad71dec27fb69faa0a57be
Author: LoGin <longjin@DragonOS.org>
Date:   Mon Aug 7 01:38:52 2023 +0800

    bugfix: 解决取消低地址映射时，错误的把重映射的物理页释放，从而导致的use after␣
↪free问题。 (#321)

commit 729a96ef47f473d535d8317a2ace5ba141fd282a
Author: Xshine <gshine@m.scnu.edu.cn>
```

(下页继续)

(续上页)

```
Date:   Sun Aug 6 12:53:47 2023 +0800

    实现内核通知链 notifier chain (#316)

    * 实现通知链块结构

    * 实现通知链的基本功能

    * 实现 atomic notifier chain

    * 实现 blocking notifier chain

    * 使用 rust 范式完成功能

    * 支持回调次数 nr_to_call

    * 移动至 libs 目录

    * 完善通知链相关方法

    * 修正相关格式

    * 文档编写

    * 更改文档路径

commit be63f3b2b6b472daa3ee17180aa607409cb9d182
Author: houmkh <1119644616@qq.com>
Date:   Sat Aug 5 18:52:46 2023 +0800

    实现系统调用Fstat (#295)

    * fstat

    * 修改syscall.rs中的verify_area

commit 9550910ae1de900e0291a84d268e8873fa142902
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Sat Aug 5 18:30:55 2023 +0800

     替换 local_irq_save 为 IrqFlagsGuard 实现 (#317)

commit abf3f634bf7e13e829556e962e7c73a85d163335
```

(下页继续)

(续上页)

```
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Aug 5 15:30:06 2023 +0800

    自动安装、更新dadk (#319)

    * auto install/auto update dadk

commit d6fd9c1e8025dd679339f9156477cb7d26d3db0d
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Aug 5 15:04:08 2023 +0800

    设置内核、relibc的远程为dragonos的git镜像站，防止国内网络问题导致编译失败 (#318)

commit 1a62e7767c1215f9668915b42de770e7993711bf
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Aug 2 18:11:05 2023 +0800

    解决relibc的头文件没能识别__dragonos__定义的问题 (#315)

commit 06500303303ec14711b4f995e2058e12703f0f2c
Author: LoGin <longjin@DragonOS.org>
Date:   Wed Aug 2 17:33:16 2023 +0800

    修正null设备以及zero设备无法open、行为不符合预期的问题 (#314)

commit 4da3758acf0327d429dfce3d313b50c2e0fc7723
Author: Chiichen <39649411+Chiichen@users.noreply.github.com>
Date:   Wed Aug 2 14:29:59 2023 +0800

    修复了wait4的异常报错 (#312)

    * 修复了wait4的异常报错

commit 821bb9a2dcfd28f9878d53ba722bdf164cf00f69
Author: Xshine <caijiaxin@dragonos.org>
Date:   Fri Jul 28 17:51:05 2023 +0800

    增加 ListenTable 来检测端口占用 (#291)

    * 增加 ListenTable 来检测端口占用


    * 使用Arc封装GlobalSocketHandle
```

(下页继续)

(续上页)

```
    * 删除 listen 处的端口检测逻辑，延至实现端口复用时完成

    * 设立两张表，分别记录TCP和UDP的端口占用

    * 实现 meatadata 相关逻辑

    * 实现socket关闭时，端口在表中移除

    * 使用端口管理器重构端口记录表

    * 修正与RawSocket相关的端口管理逻辑

    * 补充测试文件

    * 修正 unbind_port 在逻辑错误

    * 修正格式问题

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 7cc4a02c7ff7bafd798b185beb7b0c2986b9f32f
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Fri Jul 28 03:44:45 2023 -0400

    fix arch_try_cmpxchg macro declaration (#307)

    fix arch_try_cmpxchg in atomic_cmpxchg

    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>

commit a30434f5201ca4c60b9515c8c23444fea3b5a8c6
Author: zhaoyao73 <zhaoyao73@users.noreply.github.com>
Date:   Tue Jul 25 10:02:42 2023 -0400

    fix some script bugs (#304)

    add arch linux prerequisited packages

    Co-authored-by: Yao Zhao <dragonlinux@gmail.com>
```

(下页继续)

(续上页)

```
commit 40fe15e0953f989ccfeb74826d61621d43dea6bb
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Jul 22 16:27:02 2023 +0800

    新的内存管理模块 (#303)

      
↪实现了具有优秀架构设计的新的内存管理模块，对内核空间和用户空间的内存映射、分配、释放、管理等操作进

      内存管理模块主要由以下类型的组件组成：

    - **硬件抽象层（MemoryManagementArch） ** -␣
↪提供对具体处理器架构的抽象，使得内存管理模块可以在不同的处理器架构上运行
    - **页面映射器（PageMapper） **-␣
↪提供对虚拟地址和物理地址的映射，以及页表的创建、填写、销毁、权限管理等操作。分为两种类型：内核页表
    - **页面刷新器（PageFlusher） ** -␣
↪提供对页表的刷新操作（整表刷新、单页刷新、跨核心刷新）
    - **页帧分配器（FrameAllocator） ** -␣
↪提供对页帧的分配、释放、管理等操作。具体来说，包括BumpAllocator、BuddyAllocator
    - **小对象分配器** -␣
↪提供对小内存对象的分配、释放、管理等操作。指的是内核里面的SlabAllocator␣
↪（SlabAllocator的实现目前还没有完成）
    - **MMIO空间管理器** -␣
↪提供对MMIO地址空间的分配、管理操作。（目前这个模块待进一步重构）
    - **用户地址空间管理机制** - 提供对用户地址空间的管理。
        - VMA机制 - 提供对用户地址空间的管理，包括VMA的创建、销毁、权限管理等操作
        - 用户映射管理 - 与VMA机制共同作用，管理用户地址空间的映射
    - **系统调用层** -␣
↪提供对用户空间的内存管理系统调用，包括mmap、munmap、mprotect、mremap等
    - **C接口兼容层** - 提供对原有的C代码的接口，是的C代码能够正常运行。


    除上面的新增内容以外，其它的更改内容：
    - 新增二进制加载器，以及elf的解析器
    - 解决由于local_irq_save、local_irq_restore函数的汇编不规范导致影响栈行为的bug。
    - 解决local_irq_save未关中断的错误。
    - 修复sys_gettimeofday对timezone参数的处理的bug

    ---------

    Co-authored-by: kong <kongweichao@dragonos.org>

commit bb5f098a864cee36b7d2c1ab9c029c0280d94a8a
```

(下页继续)

(续上页)

```
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Jul 22 16:24:55 2023 +0800

    Revert "新的内存管理模块 (#301)" (#302)

    This reverts commit d8ad0a5e7724469abd5cc3cf271993538878033e.

commit d8ad0a5e7724469abd5cc3cf271993538878033e
Author: LoGin <longjin@DragonOS.org>
Date:   Sat Jul 22 16:22:17 2023 +0800

    新的内存管理模块 (#301)

      
↪实现了具有优秀架构设计的新的内存管理模块，对内核空间和用户空间的内存映射、分配、释放、管理等操作进

      内存管理模块主要由以下类型的组件组成：

    - **硬件抽象层（MemoryManagementArch）** -␣
↪提供对具体处理器架构的抽象，使得内存管理模块可以在不同的处理器架构上运行
    - **页面映射器（PageMapper）**-␣
↪提供对虚拟地址和物理地址的映射，以及页表的创建、填写、销毁、权限管理等操作。分为两种类型：内核页表
    - **页面刷新器（PageFlusher）** -␣
↪提供对页表的刷新操作（整表刷新、单页刷新、跨核心刷新）
    - **页帧分配器（FrameAllocator）** -␣
↪提供对页帧的分配、释放、管理等操作。具体来说，包括BumpAllocator、BuddyAllocator
    - **小对象分配器** -␣
↪提供对小内存对象的分配、释放、管理等操作。指的是内核里面的SlabAllocator␣
↪（SlabAllocator的实现目前还没有完成）
    - **MMIO空间管理器** -␣
↪提供对MMIO地址空间的分配、管理操作。（目前这个模块待进一步重构）
    - **用户地址空间管理机制** - 提供对用户地址空间的管理。
        - VMA机制 - 提供对用户地址空间的管理，包括VMA的创建、销毁、权限管理等操作
        - 用户映射管理 - 与VMA机制共同作用，管理用户地址空间的映射
    - **系统调用层** -␣
↪提供对用户空间的内存管理系统调用，包括mmap、munmap、mprotect、mremap等
    - **C接口兼容层** - 提供对原有的C代码的接口，是的C代码能够正常运行。


    除上面的新增内容以外，其它的更改内容：
    - 新增二进制加载器，以及elf的解析器
    - 解决由于local_irq_save、local_irq_restore函数的汇编不规范导致影响栈行为的bug。
    - 解决local_irq_save未关中断的错误。
```

(下页继续)

(续上页)

```
    - 修复sys_gettimeofday对timezone参数的处理的bug

commit 0663027b111ffb6ff93becd60ffef1e2b8fbd4c6
Author: TingHuang <92705854+TingSHub@users.noreply.github.com>
Date:   Wed Jul 12 12:49:45 2023 +0800

    注册串口设备，创建字符设备框架(#290)

    * 按照rust规范修改两个函数名称

    * 修改一些函数句柄以符合rust规范

    * 添加字符设备相关

    * 添加字符设备相关文件

    * 添加字符设备驱动框架代码

    * 将串口注册

    * 规范代码

commit cc36cf4a186be834e6c2ab857b9b9501ddb8b1eb
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Sat Jul 8 17:22:42 2023 +0800

    PCI设备中断重构，删去USB相关代码 (#285)

    * 修复ecam无法获取MCFG table的问题

    * 完善pcie

    * 完善irq的错误检测机制

commit 2311e2f30048d09250afc3e2e4e7029627996655
Author: 櫻井桃華 <89176634+TihayaKousaka@users.noreply.github.com>
Date:   Fri Jul 7 22:50:46 2023 +0800

    修改makefile通过编译 (#287)

commit 36fd013004ee0bd5fc7cfb452ba22531a83a859c
Author: houmkh <1119644616@qq.com>
Date:   Sat Jun 17 22:48:15 2023 +0800
```

(下页继续)

(续上页)

```
    实现gettimeofday()系统调用和clocksource+timekeeping子模块 (#278)

    - 实现gettimeofday()系统调用
    - 实现clocksource+timekeeping子模块部分功能
    - 实现了timespec转换成日期时间

commit a55ac7b928a6ca08483bbb3355bea55f1446ccab
Author: LoGin <longjin@DragonOS.org>
Date:   Tue Jun 6 17:44:54 2023 +0800

    Update issue templates (#277)

commit 5f57834372f6cb720ba14103effa4799e195a963
Author: Tptogiar <2528891112@qq.com>
Date:   Tue Jun 6 16:41:02 2023 +0800

    Makefile: 根目录下添加make help命令 (#271)

    * Makefile: 根目录下添加make help命令

    * Makefile: 补充根目录Makefile的help命令

commit aa0367d69e15989684109c5b454e85da9ecb1975
Author: LoGin <longjin@DragonOS.org>
Date:   Tue May 30 10:21:11 2023 +0800

    新的ipi功能&kick_cpu功能的重写 (#274)

commit bb24249faabc5006784aa98ca17b4cbdcb788c65
Author: LoGin <longjin@DragonOS.org>
Date:   Sun May 28 23:00:37 2023 +0800

    添加AlignBox和int_like宏 (#272)

commit ab5c8ca46db8e7d4793a9791292122b0b9684274
Author: login <longjin@DragonOS.org>
Date:   Wed May 24 17:05:33 2023 +0800

    重构系统调用模块 (#267)

    * 完成系统调用模块重构
```

(下页继续)

(续上页)

```
    * 更新github workflow

commit 660a04cef803fd73e9b294b30a96421b021a4b9b
Author: login <longjin@DragonOS.org>
Date:   Sat May 13 21:17:12 2023 +0800

    新增http server (#265)

    * 1.修复了当传入ahci驱动的缓冲区地址为用户缓冲区时,产生的内存越界问题.
↪(采用分配内核缓冲区的方式临时解决)
    2.新增http server

    * 把libssl-dev添加到bootstrap.sh

    * http_server增加对父级相对路径的安全检查，防止访问系统内的其他文件

    * 检查空指针情况

    * 解决由于链接时，crt*.o未按照升序排列导致init段链接错误的问题

commit 49249f4ec94fad7baf923aed68d9a7b2da3de3d4
Author: Bullet <93781792+GP-Bullet@users.noreply.github.com>
Date:   Sat May 13 09:55:24 2023 +0800

    把调度器实例的裸指针改为Option (#262)

commit bfafc102798ab1968ccf6b04315d8d3359a70ca8
Author: login <longjin@DragonOS.org>
Date:   Thu May 11 17:41:42 2023 +0800

    修复读取stdin时，无法正常读取的问题。 (#264)

commit 7285c927d95bb4b5c692c51a8f86c47009d07667
Author: login <longjin@DragonOS.org>
Date:   Thu May 11 16:17:58 2023 +0800

    添加dadk支持 (#263)

    * 引入dadk，使用dadk0.1.1来编译test-relibc程序

    * 由于gitee仓库体积限制导致无法继续使用gitee上的rust索引，因此更换为清华源

    * github workflow的环境中，安装dadk
```

(下页继续)

(续上页)

```
    * Auto configure dragonos rust toolchain

commit b11bb1b25676f528ec1b0e1da0af82b4652f70c4
Author: login <longjin@DragonOS.org>
Date:   Sun May 7 22:20:33 2023 +0800

    固定编译工具链、修复由于新版rust编译器问题导致的报错。 (#258)

    * 固定编译工具链、修复由于新版rust编译器问题导致的报错。

    * 完善github workflow环境配置
```

21.3 V0.1.7

备注： 本文作者：龙进 longjin@DragonOS.org

2023 年 4 月 24 日

21.3.1 贡献者名单

DragonOS V0.1.7 版本由以下小伙伴贡献代码：

- 龙进 longjin@DragonOS.org
- 关锦权 guanjinquan@DragonOS.org
- 黄厅 huangting@DragonOS.org
- 侯嘉滢 houjiaying@DragonOS.org
- 吴宇健 wuyujian@DragonOS.org
- 苏🄵韬 sujintao@DragonOS.org
- 周瀚杰 zhouhanjie@DragonOS.org
- HoshuChiu 129569557+HoshuChiu@users.noreply.github.com
- Bullet 93781792+GP-Bullet@users.noreply.github.com

21.3.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力!

- 万晓兰
- David Wen
- Seele.Clover
- FindWangHao
- ferchiel
- 叶锦毅
- 林
- Albert
- TerryLeeSCUT · GitHub
- slientbard
- 悟
- 匿名热心人士

21.3.3 更新内容-内核

- scheduler: 修改 CFSqueue 从 Vec 变成红黑树 (#229)
- new: lazy_init (#230) (#236)
- pci: pci 重构 +pcie 支持 (#235)
- net: 增加网络子系统，且能在用户态进行编程 (#237) (#247)
- mm: 调整 brk 系统调用，使得参数、返回值与 Linux 一致 (#238)
- 修改 errno, 使其与 relibc 的保持一致 (#234)
- pci: 修复 ecam 无法获取 MCFG table 的问题 (#241)
- libs: DowncastArc and its docs (#244)
- softirq: 增加定时器和软中断文档，修改了 softirq 面向 c 的接口 (#245)
- spinlock: 修复 spinlock 忘记恢复 rflags 的问题 (#247)
- waitqueue: 增加 wakeup_all 和 sleep_without_schedule 的功能 (#247)(#253)
- filesystem: 把 PollStatus 结构体改为使用 bitflags 库来实现 (#247)
- filesystem: 增加 iovec 的支持 (暴力实现) (#247)
- filesystem: 新增 SysFS (#250) (#254)

- driver: 根据 sysfs, 完善设备驱动模型 (#254)
- pipe: 匿名管道重构 (#253)
- irq: 新增 IrqArch 抽象。以及 IrqFlagsGuard。以简化关中断-恢复中断的过程 (#253)

21.3.4 更新内容-用户环境

新增仓库

- 新增子项目：dsc
- 新增子项目：DADK DragonOS Application Development Kit

DragonOS-relibc

- Add sys_dup and sys_dup2 support (#2)
- 添加原本的 libc 的内存分配器，修复对齐问题。(#6) (#7)
- 配置网络相关的系统调用 (#8)
- 修复由于 DragonOS 不支持 TLS(thread local storage) 导致 errno 变量无法正常工作的问题. (#8)

21.3.5 更新内容-其他

- build: 修复 Issue#220；vnc 的端口号恢复 5900 (#243)
- bootstrap: 解决使用 zsh 在构建 DragonOS 时，无法直接使用一键初始化脚本进行安装的问题 (#252)

21.3.6 更新内容-软件移植

无

21.3.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/DragonOS-Community/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn
- 国内镜像加速：[https://mirrors.ringotek.cn/] (https://mirrors.ringotek.cn/)

21.3.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT
- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT
- redox - https://gitlab.redox-os.org/redox-os/redox - MIT

21.3.9 当前版本的所有提交记录

```
commit e0de0fd6a52199753a3127cfbb5d12f0a1555aae
Author: TingHuang <92705854+TingSHub@users.noreply.github.com>
Date:   Sun Apr 23 22:55:57 2023 +0800

    根据sysfs完善设备驱动模型 & 添加sysfs官方文档 (#254)

    * 根据sysfs完善设备驱动模型

    * 添加sysfs官方文档

commit f678331a3315b7847f08ab32b42d5bf49a9f3a6a
Author: hanjiezhou <zhouhanjie@dragonos.org>
Date:   Sun Apr 23 21:05:10 2023 +0800

    匿名管道重构&增加IrqArch trait以及IrqFlags及其守卫 (#253)

    * 实现匿名管道
```

(下页继续)

(续上页)

```
    * 增加IrqArch trait以及IrqFlags及其守卫

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 8a1e95abb5e4df5e872bb452efc26c9e9631157d
Author: Bullet <93781792+GP-Bullet@users.noreply.github.com>
Date:   Fri Apr 21 23:36:54 2023 +0800

    解决使用zsh在构建DragonOS时，无法直接使用一键初始化脚本进行安装的问题 (#252)

commit dd9f1fc1a42406461e6f0d38cce1e56e22a1a15f
Author: TingHuang <92705854+TingSHub@users.noreply.github.com>
Date:   Fri Apr 21 16:03:42 2023 +0800

    新增SysFS (#250)

    * 添加sysfs

    * 注册sysfs

    * 添加sysfs相关

    * 添加rust-anlyzer辅助配置

    * 将设备与sysfs相关联

    * 添加单独的文件管理sysfs下的文件夹

commit cde5492f725681ed89abe1e6eb088e05d943d793
Author: login <longjin@DragonOS.org>
Date:   Wed Apr 19 18:05:02 2023 +0800

    新增网络socket的系统调用接口 (#247)

    1.修复spinlock忘记恢复rflags的问题
    2.WaitQueue增加wakeup_all的功能
    3.完善tcp,udp,raw socket
    4.把PollStatus结构体改为使用bitflags
    5.新增iovec结构体
    6.完成网络的系统调用
    7.在bootstrap里面添加dnsmasq bridge-utils iptables
```

(下页继续)

(续上页)

```
    ---------

    Co-authored-by: guanjinquan <1666320330@qq.com>

commit 8fd71f277271ae68e648f290c67f187b030feae0
Author: houmkh <1119644616@qq.com>
Date:   Mon Apr 17 17:17:06 2023 +0800

    增加定时器和软中断文档，修改了softirq面向c的接口 (#245)

    * 增加定时器和软中断文档

    * 修改softirq对c的接口和文档

    * 修改文档格式

commit 77c928f6ce3192c79ea42ab7bcba2713e289f73b
Author: login <longjin@DragonOS.org>
Date:   Sun Apr 16 20:29:04 2023 +0800

    new: DowncastArc and its docs (#244)

commit 7149abaa49a4ca70f0e42ad3b61fdfd6a941a092
Author: HoshuChiu <129569557+HoshuChiu@users.noreply.github.com>
Date:   Sun Apr 16 14:47:51 2023 +0800

    修复Issue#220; vnc的端口号恢复5900 (#243)


    * 修复Issue#220

    * qemu-vnc端口号恢复为5900

commit 5c1e552cc7f0a6ad75c8a1fa2928e3b9cc619657
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Fri Apr 14 12:21:08 2023 +0800

    修复ecam无法获取MCFG table的问题 (#241)

commit 79a452ce8f27ad9c7283ac0bcf4078ed6fa018d7
Author: houmkh <1119644616@qq.com>
Date:   Tue Apr 11 17:05:33 2023 +0800
```

(下页继续)

(续上页)

```
    修改errno,使其与relibc的保持一致 (#234)

    修改errno,使其与relibc的保持一致

commit ac48398d3f17f24ff9b5da5e400ce912d05f0ba2
Author: login <longjin@DragonOS.org>
Date:   Tue Apr 11 16:54:14 2023 +0800

    调整brk系统调用，使得参数、返回值与Linux一致 (#238)

    * 新增用于测试relibc的app

    * 为适配relibc,修改do_execve中关于用户栈的内容的设置

    * 调整brk系统调用，使得参数、返回值与Linux一致

commit 13776c114b15c406b1e0aaeeb71812ea6e471d2e
Author: login <longjin@DragonOS.org>
Date:   Mon Apr 10 20:22:39 2023 +0800

    增加对dhcpv4的支持（tcp、udp socket已写好，但由于缺少epoll机制，尚未完整测试） (
↪#237)

    * 为virtio网卡完成smoltcp的phy层配置

    * raw socket

    * 初步写完udp和tcp socket

    * 能够正常通过dhcp获取ipv4地址（具有全局iface btree)

    ---------

    Co-authored-by: guanjinquan <1666320330@qq.com>

commit 78bf93f02f84bf5e024ddfb559f040e68ce39ccf
Author: YJwu2023 <yujianwu2019@gmail.com>
Date:   Sun Apr 9 12:30:02 2023 +0800

    pci重构+pcie支持 (#235)

    * pci重构+pcie支持
```

(下页继续)

(续上页)

```
    * pci重构测试完成

    * 修正makefile的问题

    * 小修改

    * 修改函数名字

commit 5c9a63df836eedaca33c8c4c600b7aaeb2caf9a6
Author: login <longjin@DragonOS.org>
Date:   Sat Apr 8 23:53:53 2023 +0800

    Patch add lazy init (#236)

    * 修正并发安全问题

commit 766127209ee49465a8086cfd0bec90d8b79a96c0
Author: login <longjin@DragonOS.org>
Date:   Thu Apr 6 19:01:30 2023 +0800

    new: lazy_init (#230)

commit e0dfd4d5d70d1b50fc7ad3ed4bf84b7ba6dad19d
Author: hanjiezhou <zhouhanjie@dragonos.org>
Date:   Thu Apr 6 00:50:14 2023 +0800

    修改CFSqueue从Vec变成红黑树 (#229)

    使用了由tickbh编写的rbtree： https://github.com/tickbh/rbtree-rs/blob/master/src/
↪lib.rs

    Co-authored-by: tickbh <tickdream125@hotmail.com>

commit 2a7d773d3d39f1cb3d59d6baa817c896c6fd52d1
Author: TingHuang <92705854+TingSHub@users.noreply.github.com>
Date:   Wed Apr 5 13:02:05 2023 +0800

    新增设备驱动模型，为设备和驱动提供高层视图 (#227)

    * 添加base mod

    * 添加设备驱动模型相关文件
```

(下页继续)

(续上页)

```
    * 删除单独的mod文件，使用mod.rs，修改一些格式上的问题

    * 移动驱动错误类型到该文件

    * 修改一些格式上的问题

commit 5d00b1852818dd4b25952fd6a30deb20e7c7df53
Author: login <longjin@DragonOS.org>
Date:   Wed Apr 5 00:53:35 2023 +0800

    修复显示刷新线程的空指针问题 (#228)
```

21.4 V0.1.6

备注： 本文作者：龙进 longjin@DragonOS.org

2023 年 4 月 2 日

21.4.1 贡献者名单

DragonOS V0.1.6 版本由以下小伙伴贡献代码：

- 龙进 longjin@DragonOS.org
- 关锦权 guanjinquan@DragonOS.org
- 苏🄵韬 sujintao@DragonOS.org
- 侯嘉滢 houjiaying@DragonOS.org
- 吴宇健 wuyujian@DragonOS.org
- Mork 91721145+MorkCarpenter@users.noreply.github.com
- WaferJay 17383312+WaferJay@users.noreply.github.com
- HoshuChiu 129569557+HoshuChiu@users.noreply.github.com

21.4.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- 万晓兰
- David Wen
- Seele.Clover
- FindWangHao
- 叶锦毅
- 林
- Albert
- TerryLeeSCUT · GitHub
- slientbard
- 悟
- 匿名热心人士

21.4.3 更新内容-内核

- softirq: 重构了软中断 (#223)
- timer: 重构了系统定时器 (#223)
- stdio: 新增 tty 设备，用于标准输入输出 (#202) (#217)
- lib: 第一套键盘扫描码的状态机 (#216) (#219)
- syscall: 新增 dup,dup2 系统调用 (#224)
- syscall: 新增 SystemError 枚举类型，使得错误处理更清晰 (#205)
- driver: 新增 x87 浮点处理器支持 (#212)
- driver: VirtIO 网卡能够正常发送、接收数据 (#204)
- filesystem: 修正了 FAT32 判断逻辑，解决了文件系统为 FAT12/16 时系统无法正常启动的问题。(#211)
- filesystem: 新增 VFS 文档，以及修改文档配置 (#209)
- textui: 修复由于 textui 加锁，更改了 preempt_count 导致“进程长时间连续输出字符”的情况下，进程调度器不运行的问题。(#203)
- scheduler: 解决由于在中断上下文以外，sched_enqueue 时，未关中断导致 cpu_queue 双重加锁的问题 (#201)

21.4.4 更新内容-用户环境

新增仓库

- 新增子项目：dsc
- 移植 relibc: DragonOS-relibc

21.4.5 更新内容-其他

- build: 添加了 qemu 使用 VNC 作为图像输出的选项 (#222)

21.4.6 更新内容-软件移植

无

21.4.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/DragonOS-Community/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn
- 国内镜像加速：[https://mirrors.ringotek.cn/] (https://mirrors.ringotek.cn/)

21.4.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT

- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT
- redox - https://gitlab.redox-os.org/redox-os/redox - MIT

21.4.9 当前版本的所有提交记录

```
commit bacd691c9ef0502b5cc618aad50517f9e59df5e0
Author: login <longjin@DragonOS.org>
Date:   Sun Apr 2 17:09:33 2023 +0800

    软中断&定时器重构 (#223)

    * 软中断&定时器重构

    Co-authored-by: houmkh<houjiaying@DragonOS.org>

    * 修改timer的clock()

    * 删除debug信息

    ---------

    Co-authored-by: houmkh <1119644616@qq.com>

commit 6d345b774223b0daaf0ee629c7fb595a1912a9e2
Author: HoshuChiu <129569557+HoshuChiu@users.noreply.github.com>
Date:   Sun Apr 2 15:55:24 2023 +0800

    添加了qemu使用VNC作为图像输出的选项 (#222)

    * 添加了qemu使用VNC作为图像输出的选项

    * 设置vnc端口为5900

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 2b771e32f5795e0fdda458e3bb2651ef6b9673ac
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Sun Apr 2 15:43:53 2023 +0800

    Add dup,dup2 (#224)
```

(下页继续)

(续上页)

```
    * dup,dup2

    * fix: sys_dup2语义与posix不一致的问题

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit d7b31a969ff091224a4929496f0278d024f78c77
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Fri Mar 31 18:23:58 2023 +0800

    Patch keyboard capslock alt (#219)

    * keyboard-alt-capslock

    * 解决键盘输入'%'字符的时候无法回显的bug

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 20e3152e1eea97f87d644c3023391e172bc83c93
Author: login <longjin@DragonOS.org>
Date:   Fri Mar 31 12:54:37 2023 +0800

    将TTY与stdio进行连接,实现基本的stdio功能 (#217)

    * 将stdio与tty接上

commit 5fb12ce447710edf8566f250655a06cb27519fca
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Thu Mar 30 18:19:02 2023 +0800

    第一套键盘扫描码的状态机 (#216)

    第一套键盘扫描码的状态机
    ---------

    Co-authored-by: guanjinquan <1666320330@qq.com>
    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 676b8ef62e1a0a1e52d65b40c53c1636a2954040
```

(下页继续)

(续上页)

```
Author: Mork <91721145+MorkCarpenter@users.noreply.github.com>
Date:   Wed Mar 29 21:24:11 2023 +0800

    部分函数从返回值为Result<>,i32>修改为Result<>,SystemError> (#210)

    * 将Result<>,i32>替换为Result<>,SystemError>
    * bugfix: 显示双缓冲区初始化的时候，连续注册了两次Video Softirq的问题。

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 64aea4b3494bee7375e1c1ee5739c9fab0db0cb7
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Tue Mar 28 20:44:26 2023 +0800

    增加x87FPU支持 (#212)

    * remove `ret_from_syscall`
    *修复ps2键盘驱动程序inode在进程fork的时候导致死锁的问题.
    *更新: VFS每次拷贝文件描述符的时候,都会去调用inode的open函数

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 2286eda6526ed1b46afd79b47dc701034b9e903d
Author: WaferJay <17383312+WaferJay@users.noreply.github.com>
Date:   Mon Mar 27 09:32:43 2023 +0800

    修正了FAT32判断逻辑，解决了文件系统为FAT12/16时系统无法正常启动的问题。 (#211)

    * fix(fat): fix determination of fat type casue crash if fs is fat12/16

    * refactor(fat): split BiosParameterBlock.validate() into BiosParameterBlockFAT32.
↪validate() and BiosParameterBlockLegacy.validate()

    * 调整“最大允许的簇号”的常量放置的位置。

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 45b8371173b070028457f7ee64be33f68b4f9ada
Author: login <longjin@DragonOS.org>
```

(下页继续)

(续上页)

```
Date:   Sat Mar 25 14:51:16 2023 +0800

    新增VFS文档，以及修改文档配置 (#209)

    * 1.新增vfs设计文档
    2.修改文档版权标志为"2022-2023, DragonOS Community"
    3.修改电脑版文档页面的宽度为90%

    * layout.html末尾加空行

commit 73c607aaddf6e4634cad179a81d3f1bc589f7220
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Sat Mar 18 20:43:37 2023 +0800

    VirtIO网卡能够正常发送、接收数据 (#204)

    * virtio-net小修改

    * 移动volatile.rs到libs文件夹

    * 使用virtio-drivers 0.3.0

    * bugfix: 初始化BAR之后，未正确设置command register的问题


    ---------

    Co-authored-by: longjin <longjin@dragonos.org>

commit 4454d1a2dd1f1078750151c028a794cfd9a04a1b
Author: login <longjin@DragonOS.org>
Date:   Sat Mar 18 20:26:05 2023 +0800

    新增SystemError枚举类型，使得错误处理更清晰 (#205)

commit 0d48c3c9c21a2dd470d0e1e58b507db60e0887bb
Author: login <longjin@DragonOS.org>
Date:   Thu Mar 16 19:48:59 2023 +0800

    new: tty设备（尚未与stdio接上） (#202)

commit 790d45764090bce3bbfb96b42b2818100a8cef9a
Author: login <longjin@DragonOS.org>
```

(下页继续)

(续上页)

```
Date:   Wed Mar 15 11:42:41 2023 +0800

    修复由于textui加锁，更改了preempt_
↪count导致“进程长时间连续输出字符”的情况下，进程调度器不运行的问题。 (#203)

commit c2e757d8cbeed01b16f48bea48ed8447685e6f1a
Author: login <longjin@DragonOS.org>
Date:   Mon Mar 13 22:22:23 2023 +0800

    解决由于在中断上下文以外，sched_enqueue时，未关中断导致cpu_queue双重加锁的问题 (
↪#201)
```

21.5 V0.1.5

备注： 本文作者：龙进 longjin@RinGoTek.cn

2023 年 3 月 13 日

21.5.1 贡献者名单

DragonOS V0.1.5 版本由以下小伙伴贡献代码：

- 龙进 longjin@DragonOS.org
- 关锦权 guanjinquan@DragonOS.org
- 苏🄵韬 sujintao@DragonOS.org
- 孔维超 kongweichao@DragonOS.org
- 侯嘉滢 houjiaying@DragonOS.org
- 吴宇健 wuyujian@DragonOS.org
- 周于🄵 zhouyuzhe@DragonOS.org
- Satin Wuker 74630829+SatinWuker@users.noreply.github.com

21.5.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力!

- 万晓兰
- David Wen
- Seele.Clover
- FindWangHao
- 叶锦毅
- 林
- Albert
- TerryLeeSCUT · GitHub
- slientbard
- 悟

21.5.3 更新内容-内核

- scheduler: doc: 实时进程调度器文档 (#163)
- scheduler: rt: RTQueue 改用双向链表存储 (#174)
- scheduler: load balance: 多核负载均衡 (#193)
- Semaphore: new: 新增了 rust 实现的信号量 (#183)
- mm: refactor: 重构了 MMIO 地址分配器 (#184)
- RwLock: new: 新增了 rust 实现的读写锁 (#186)
- driver: update: 完善 pci 的功能 (#194)
- driver: new: VirtIO 网卡驱动（仍存在问题）(#194)
- driver: refactor: Rust 版本的 AHCI 驱动 (#198)
- block io: delete: 移除 Block IO 调度器. (#196)
- filesystem: refactor: 新版的 VFS (#198)
- filesystem: refactor: 新版的 ProcFS (#198)
- filesystem: refactor: 新版的 DevS (#198)
- filesystem: new: RamFS 内存文件系统 (#198)
- filesystem: new: FAT12/FAT16/FAT32 文件系统 (#198)
- filesystem: new: 新的设备、块设备抽象 (#198)

21.5.4 更新内容-用户环境

- libc: 调整，将所有的 app 直接链接到 libc.a 中，而不是都执行一遍”搜索.o”的过程 (#171)

21.5.5 更新内容-其他

- bootstrap: 解决 ubuntu2210 版本无法正确编译 grub，以及正确安装 qemu 的问题 (#176)
- toolchain: 添加 rust 的 bare bone 工具链 (#197)

21.5.6 更新内容-软件移植

无

21.5.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/DragonOS-Community/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn
- 国内镜像加速：[https://mirrors.ringotek.cn/] (https://mirrors.ringotek.cn/)

21.5.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT

- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT
- redox - https://gitlab.redox-os.org/redox-os/redox - MIT

21.5.9 当前版本的所有提交记录

```
commit 84407d360511c7699938a0f245ae33ff76f16b17
Author: login <longjin@DragonOS.org>
Date:   Mon Mar 13 00:26:04 2023 +0800

    bugfix: 解决touch命令失败的问题 (#199)

    * bug fix : 解决touch命令失败的问题

commit 004e86ff19727df303c23b42c7a271b9214c6898
Author: login <longjin@DragonOS.org>
Date:   Sun Mar 12 22:36:11 2023 +0800

    新版文件系统重构完成 (#198)

    1.重构: VFS
    2. 重构: ProcFS
    3. 重构: DevFS
    4. 重构: FAT32
    5. 重构: AHCI驱动
    6. 新增: RamFS
    7. 新增: MountFS
    8. 新增: FAT12
    9. 新增: FAT16
    10. 重构: 设备抽象

    Co-authored-by: guanjinquan <1666320330@qq.com>
    Co-authored-by: DaJiYuQia <88259094+DaJiYuQia@users.noreply.github.com>

commit 17041e0e307eaf9e8d8ddbddfa186cd1f10f1bc0
Author: login <longjin@DragonOS.org>
Date:   Sun Mar 12 21:04:37 2023 +0800

    添加rust的bare bone工具链 (#197)

commit 26d84a31393c50063ff416bc509316e8d342028c
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Sat Mar 11 21:09:50 2023 +0800
```

(下页继续)

(续上页)

```
    新增VirtIO网卡驱动 (#194)

    * 修复内存bug与grub安装脚本的错误

    * 修改小bug

    * PCI增加功能与virtio-net驱动

    * little fix

    * virtio-net小修改

commit 1d48996375149279a721777b2c600e1b5c3ee1b5
Author: kong <45937622+kkkkkong@users.noreply.github.com>
Date:   Sat Mar 11 18:17:35 2023 +0800

    多核负载均衡(#193)

    * feat(sched)： CPU负载检测初步实现

    * fix(smp):调整smp中的apic的头文件声明

    * fix(smp):简单的负载均衡算法实现

    * fix(sched):抽离负载均衡方法

    * fix(sched):修改rt中的运行队列bug, 调整负载均衡逻辑

    * fix(process):移除无用测试代码

    * reformat code

commit ef9f9732b09f78d7192f1d0dd3b41be655fb0914
Author: houmkh <100781004+houmkh@users.noreply.github.com>
Date:   Thu Mar 9 23:31:25 2023 +0800

    修复了mmio buddy的bug (#189)

    * 修改buddy_query

commit c1396d277115b371d09ad6d39a1c419f9224ffd0
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Mon Mar 6 11:28:32 2023 +0800
```

(下页继续)

(续上页)

```
    Rwlock文档 (#186)

    * Rwlock文档

commit a7eb62a47a8d701b90a14f83cc9028cfed07c268
Author: houmkh <100781004+houmkh@users.noreply.github.com>
Date:   Mon Mar 6 11:21:29 2023 +0800

    修改mmio - buddy代码结构和函数名 (#184)

    * 修改mmio-buddy结构和函数名

commit c2481452f81750ec02adec627ab2edbc93d9cd9c
Author: houmkh <100781004+houmkh@users.noreply.github.com>
Date:   Sat Mar 4 18:36:55 2023 +0800

    rust重构mmio_buddy和mmio (#178)

    * rust重构mmio_buddy和mmio

    * mmio-buddy文档

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit f1284c35717a2f9f8cee7cecfc835ba1d23a1161
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Sat Mar 4 17:47:17 2023 +0800

    新增了rust实现的信号量 (#181)

    * 新增了rust实现的信号量

    ---------

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 83b9512c1c1e8289000084adcafddebee6a23f16
Author: Gou Ngai <sujintao@dragonos.org>
Date:   Sat Mar 4 16:54:42 2023 +0800
```

(下页继续)

(续上页)

```
    新增了rust实现的信号量 (#183)

    * 新增了rust实现的信号量

commit e532a536a0b244f4590e6eb7910084bd63049704
Author: login <longjin@ringotek.cn>
Date:   Thu Mar 2 22:50:07 2023 +0800

    添加赞助者：FengWangHao (#179)

commit b66beefd4e9ead61ee55f335246ebeb8277d3011
Author: login <longjin@ringotek.cn>
Date:   Mon Feb 27 01:00:35 2023 +0800

    解决ubuntu2210版本无法正确编译grub，以及正确安装qemu的问题 (#176)

commit 4177d0327c3eacdc606f0b22f99f208fd48cfff3
Author: kong <45937622+kkkkkong@users.noreply.github.com>
Date:   Mon Feb 20 17:03:37 2023 +0800

    RTQueue改用双向链表存储(#174)

    * RTQueue改用双向链表存储

commit 2bf5ee0e3cac3a91dee6a13b71c86a9477c07d9b
Author: login <longjin@ringotek.cn>
Date:   Sat Feb 11 13:04:24 2023 +0800

    修改libc的编译相关内容(#171)

    1.将libc的include文件夹分为export和internal
    2.将所有app都直接链接libc.a,而不是都执行一遍"搜索.o"的过程

commit 90b077f9d3ecd48ca46f8bbb32363620db6ddbe6
Author: kong <45937622+kkkkkong@users.noreply.github.com>
Date:   Thu Feb 9 15:24:37 2023 +0800

    Sched rt doc (#163)

    * update

    * 完善调度器文档
```

(下页继续)

(续上页)

```
    * 更新RT调度器文档

    * 更新实时调度文档

commit 009f92d50fe2e52e425bce397801d3fa204daecd
Author: Satin Wuker <74630829+SatinWuker@users.noreply.github.com>
Date:   Tue Feb 7 19:29:09 2023 -0800

    fix typos 改正README_EN的错别字和语法错误 (#167)
```

21.6 V0.1.4

备注： 本文作者：龙进 longjin@RinGoTek.cn

2023 年 2 月 4 日

21.6.1 贡献者名单

DragonOS V0.1.4 版本由以下小伙伴贡献代码：

- 龙进 longjin@RinGoTek.cn
- Gou Ngai sujintao@DragonOS.org
- 孔维超 kongweichao@DragonOS.org
- 侯嘉滢 houjiaying@DragonOS.org

21.6.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- David Wen (人民币 2000 元)
- Seele.Clover (人民币 500 元)
- 叶锦毅 (人民币 100 元)
- 林 (人民币 50 元)
- Albert (人民币 9.99 元)
- TerryLeeSCUT (人民币 6.66 元)
- slientbard (人民币 6.66 元)

- 悟 (人民币 2.00 元)
- 【其他匿名的热心人士】（人民币 1.00 元）

21.6.3 更新内容-内核

- Spinlock: new: 新增具有守卫的自旋锁 SpinLock，支持编译期对锁的使用进行检查。(#148)
- Spinlock: feature: Raw spin lock 增加 lock_irqsave、unlock_irqrestore(#151)
- Mutex: new: Rust 版本的 Mutex (#157)
- doc: new: Rust 代码风格文档 (#161)
- WaitQueue: new: Rust 版本的 WaitQueue (#162)
- WaitQueue: update: C 版本的 wait_queue 的唤醒，改为立即唤醒 (#158)
- block io: new: Block IO 调度器. 当具有多核时，io 调度器在核心 1 上运行。(#158)
- smp: bugfix: 为 AP 核启动 apic_timer, 使其能够运行调度 (#158)
- smp: new: 增加 kick_cpu 功能，支持让某个特定核心立即运行调度器 (#158)
- smp: new: 增加进程在核心间迁移的功能 (#158)
- scheduler: new: 增加实时进程调度器（支持 FIFO、RR 策略）(#139)
- scheduler: update: CFS 调度器为每个核心设置单独的 IDLE 进程 pcb（pid 均为 0）(#158)
- scheduler: bugfix: process_wakeup 时，对 cfs 的进程，重设虚拟运行时间。解决由于休眠的进程，其虚拟运行时间过小，导致其他进程饥饿的问题 (#158)
- process: new: pcb 中增加 migrate_to 字段 (#158)

21.6.4 更新内容-用户环境

无

21.6.5 更新内容-其他

无

21.6.6 更新内容-软件移植

无

21.6.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/fslongjin/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn
- 国内镜像加速：[https://mirrors.ringotek.cn/] (https://mirrors.ringotek.cn/)

21.6.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。(这是最容易违反我们的开源协议的场景)。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT
- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT

21.6.9 当前版本的所有提交记录

```
commit f6ba114bb0420e848ef7fc844c96c0d7a0552d93
Author: houmkh <100781004+houmkh@users.noreply.github.com>
Date:   Sat Feb 4 12:31:15 2023 +0800

    Block IO Scheduler (#158)

    * Block io调度器
    * process_
↪wakeup时，对cfs的进程，重设虚拟运行时间。解决由于休眠的进程，其虚拟运行时间过小，导致其他进程饥饿的
```

(下页继续)

(续上页)

```
    * 1、为AP核启动apic_timer,使其能够运行调度
    2、增加kick_cpu功能，支持让某个特定核心立即运行调度器
    3、wait_queue的唤醒，改为立即唤醒。
    4、增加进程在核心间迁移的功能
    5、CFS调度器为每个核心设置单独的IDLE进程pcb（pid均为0）
    6、pcb中增加migrate_to字段
    7、当具有多核时，io调度器在核心1上运行。

    * io调度器文件位置修改

    * 修改io的makefile

    * 更新makefile中的变量名

    * 修改io调度器函数名

    ---------

    Co-authored-by: login <longjin@ringotek.cn>

commit 151251b50b7ed55596edd32ffec49a4041010e2a
Author: login <longjin@ringotek.cn>
Date:   Tue Jan 31 19:27:02 2023 +0800

    Patch add rust waitqueue (#162)

    * new: rust版本的waitqueue

    * new:等待队列的文档

commit 3c369b1430e8d571bcc74a8ef7fefc1c4cae5dd2
Author: login <longjin@ringotek.cn>
Date:   Mon Jan 30 15:43:42 2023 +0800

    new:新增rust代码风格 (#161)

commit c28bd540ac856cd9d8d5597852af8f2588a660e4
Author: login <longjin@ringotek.cn>
Date:   Mon Jan 30 15:10:24 2023 +0800

    更新赞助者名单 (#160)
```

(下页继续)

(续上页)

```
    * 更新赞赏者列表

commit 935f40ec174fec217aed4553d45996327443bc0e
Author: login <longjin@ringotek.cn>
Date:   Tue Jan 17 21:30:16 2023 +0800

    new: Rust版本的Mutex (#157)

commit d8a064128a8a06b90ff4c7b87c193518d9572641
Author: Gou Ngai <94795048+AlbertSanoe@users.noreply.github.com>
Date:   Mon Jan 16 19:58:50 2023 +0800

    Raw spin lock 增加lock_irqsave、unlock_irqrestore(#151)

    Raw spin lock 增加lock_irqsave、unlock_irqrestore

commit 06b09f34ed64a006a80ae8df383e3c8b176f02e0
Author: kong <45937622+kkkkkong@users.noreply.github.com>
Date:   Sat Jan 14 22:38:05 2023 +0800

    Patch sched rust (#139)

    * update

    * 添加rt调度器的rust初步实现

    * 完善rt调度逻辑

    * 调试rt调度器

    * 修改sched的返回值

    * cargo fmt 格式化

    * 删除无用代码，修补rt bug

    * 删除无用的代码，和重复的逻辑

    * 软中断bugfix

    * 删除一些代码

    * 添加kthread_run_rt文档
```

(下页继续)

(续上页)

```
    * 解决sphinix警告_static目录不存在的问题

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit ec53d23ed03347854189d92b7e175f309779321b
Author: login <longjin@ringotek.cn>
Date:   Sat Jan 14 10:35:49 2023 +0800

    new: 新增具有守卫的自旋锁SpinLock，支持编译期对锁的使用进行检查。 (#148)

commit 41474ba3df99b6822ce452dc94dc53a4da62cba1
Author: login <longjin@ringotek.cn>
Date:   Tue Jan 10 22:07:41 2023 +0800

    更新Readme中关于DragonOS的介绍部分 (#146)

commit 8ad2e358fd3b05eed2919de50640682e51687fb5
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 15:51:59 2023 +0800

    更新about app中的版本号 (#145)

    * 更新about app中的版本号

commit a8b621c8d1fe77251b8e4eafe258dc0ee7366dd5
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 15:47:44 2023 +0800

    修正由于libc中具有crti.S和crtn.S，造成的与x86_64-elf-gcc不兼容的问题 (#144)

commit 9358ff0f6f7daa18d6fab4497de025736b3d6725
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 15:06:52 2023 +0800

    Add v0.1.3 changelog (#143)

    * new: 0.1.3发行日志

    * 新增输出指定时间范围内的贡献者名单的脚本

    * 更新bootloader文档
```

(下页继续)

(续上页)

```
    * update: 简介文档

    * new: 镜像站文档

    * update: 功能特性文档
```

21.7 V0.1.3

备注： 本文作者：龙进 longjin@RinGoTek.cn

2023 年 1 月 8 日

21.7.1 贡献者名单

DragonOS V0.1.3 版本由以下小伙伴贡献代码：

- 龙进 longjin@RinGoTek.cn
- 吴宇健 wuyujian@DragonOS.org
- 关锦权 guanjinquan@DragonOS.org
- Gou Ngai sujintao@DragonOS.org

21.7.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- David Wen
- Seele.Clover
- TerryLeeSCUT
- 悟
- slientbard

21.7.3 更新内容-内核

- syscall: new: 增加 getpid 系统调用 (#120)
- signal: update: 对于除了 SIGKILL 以外的信号，也将他们加入 SigQueue (#120)
- rtc: refactor: 使用 Rust 重构 RTC 驱动 (#118)
- doc: new: 新增 signal 的文档 (#126)
- Spinlock: new: 新增 rust 写的 RawSpinlock (#127)
- arch: update: 在 lib.rs 中，将 arch 模块的路径进行更改，使得其他模块使用 arch 的代码时，不需要指定 arch::x86_64 (#128)
- mm: bugfix: 修复页面分配器在初始化时，ZONE_NORMAL_INDEX 始终为 0 的 bug (#129)
- scheduler: new: 使用 Rust 重构 CFS 调度器 (#131)
- smp: 删除已经在 smp 中废弃的 HPET 中断转发函数 (#131)
- process: bugfix: 修复 init 进程忘记设定 fs gs 寄存器的问题。(#132)
- vfs: update: 将 VFS 文件夹重命名为 vfs (#133)
- lockref: new: 新增 rust 版本的 lockref (#135)
- cpu: new: new:Rust 封装 cpu_relax(), 通过 pause 指令，让 cpu 休息一会儿, 降低空转功耗. (#135)
- 使用 rust 重构 softirq 机制 (#138)

21.7.4 更新内容-用户环境

- libc: bugfix: 注册信号处理函数时，总是注册 sigkill 的问题 (#120)
- libc: new: 增加了 raise、kill、abort (#120)
- libc: new: 新增 arch 文件夹，在下面新增 crt0 crti crtn 文件 (#134)
- libc: new: 新增 `fflush()`, `fprintf()`, `stdin`, `stdout`, `stderr`, `ferror()`, `fopen()`, `fclose()`, `putchar()`, `puts()` (#136)
- libc: new: 简单添加了 `fopen()` 对 mode 参数的处理。请注意，它没有完全遵循 posix，也与 Linux 的不一致，将来使用 Rust 的时候完善它。(#141)
- 移植: new: 新增了 gmp, mpfr, mpc 的移植构建脚本 (#136)
- 移植: new: 新增了 gcc、binutils 的交叉编译构建脚本以及 gcc-11.3.0, binutils-2.38 的补丁（在 DragonOS-community 下的仓库中）(#136)
- compile: update: 更改编译器的 Include 路径，使得 include 时不需要加`<libc/src/include/>` 前缀 (#124)

21.7.5 更新内容-其他

- bugfix: 修复 docker 安装时异常退出的 bug (#116)
- new: 新增目标为 x86_64-elf 的 GCC 裸机编译器，并使用它来编译 DragonOS (#111)
- update: 更新 Docker 编译镜像至版本 `dragonos/dragonos-dev:v1.2`, 并支持从 Dockerfile 构建这个编译镜像 (#111)
- bugfix: 修复 MBR 磁盘镜像未设置启动标志的 bug (#111)
- update: 更新 github workflow，增加 cache，加快 build check 的速度
- bugfix: 修复下载 grub2.06 时的提示错误 (#125)

21.7.6 更新内容-软件移植

- new: gcc 11.3.0 userland cross-compiler，提交：`64a5b1cbf28e3305560e166c1b6624e99745c720`, 仓库：https://github.com/DragonOS-Community/gcc
- new: binutils 2.38 cross-compile tools，提交：`a0ae560e0065862a9867b9e1f8364749ef38d99e`, 仓库：https://github.com/DragonOS-Community/binutils
- new: gmp 6.2.1, 提交：`dd9eee5778fb6027fafa4fe850aff21b1a71c18e`, 仓库：https://github.com/DragonOS-Community/gmp-6.2.1
- new: mpfr 4.1.1, 提交：`fa8e30cdc2e838fdd82b60fec31fcfc5e118aad6`, 仓库：https://github.com/DragonOS-Community/mpfr
- new: mpc 1.2.1,（无需打补丁即可移植），仓库：https://github.com/DragonOS-Community/mpc

21.7.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/fslongjin/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn
- 国内镜像加速：[https://mirrors.ringotek.cn/] (https://mirrors.ringotek.cn/)

21.7.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT
- rcore-fs - https://github.com/rcore-os/rcore-fs.git - MIT

21.7.9 当前版本的所有提交记录

```
commit a8b621c8d1fe77251b8e4eafe258dc0ee7366dd5
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 15:47:44 2023 +0800

    修正由于libc中具有crti.S和crtn.S，造成的与x86_64-elf-gcc不兼容的问题 (#144)

commit 9358ff0f6f7daa18d6fab4497de025736b3d6725
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 15:06:52 2023 +0800

    Add v0.1.3 changelog (#143)

    * new: 0.1.3发行日志

    * 新增输出指定时间范围内的贡献者名单的脚本

    * 更新bootloader文档

    * update: 简介文档
```

(下页继续)

(续上页)

```
    * new: 镜像站文档

    * update: 功能特性文档

commit fd91905f022b3ceaa59e666d1ff42d91fb8d40ef
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 8 11:38:59 2023 +0800

    解决编译gcc、binutils的脚本中，变量名称错误的问题 (#142)

commit 62e4613978193aaf5d949a331df0398f2d085a30
Author: Gou Ngai <94795048+AlbertSanoe@users.noreply.github.com>
Date:   Sat Jan 7 23:15:37 2023 +0800

    使用rust重构softirq机制；解决Rtc驱动的编译警告问题 (#138)

    * 使用rust重构softirq机制
    * 解决Rtc驱动的编译警告问题

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit e9fdc57bf878f1bc5cc5743dfaeeaef743439291
Author: login <longjin@ringotek.cn>
Date:   Sat Jan 7 22:36:49 2023 +0800

    ␣
↪简单添加了fopen对mode参数的处理。请注意，它没有完全遵循posix，也与Linux的不一致，将来使用Rust的时候
↪(#141)

commit 2224c93ea968bc74621f7e124b4aca04875b3e6a
Author: guanjinquan <1666320330@qq.com>
Date:   Fri Jan 6 21:29:23 2023 +0800

    完善libc，构建了OS-specific工具链，编译了基于gcc-11.3.0的DragonOS userland␣
↪compiler，移植了mpfr,gmp,mpc库 (#134)

    * 修改include路径

    * 添加了创建libsysapi.a和/bin/sysroot/usr/include/+lib/的代码

    * 修补.gitignore

    * 删除多余项
```

(下页继续)

(续上页)

```
* 优化脚本可读性

* 新增crt0 crti crtn

* 编译binutils所需的东西

* fflush()和fprintf()的简单实现

* 应用程序启动前，调用初始化libc的函数

* 自动创建sysroot

* 添加了stderr的初始化

* 修改了stderr的初始化

* 内核添加对stdio的简略处理

* 格式化代码

* 修正打开stdio文件描述符的问题

* bugfix: 修复fprintf忘记释放buf的问题

* 修复shell错误地把入口设置为main而不是_start的问题

* 新增__cxa_atexit  (gcc要求libc提供这个)

* 增加putchar puts

* 更新写入磁盘镜像的脚本，默认无参数时，使用legacy方式安装

* 更新编译脚本

* stdio增加eof的定义

* 新增extern cplusplus

* mpfr gmp mpc 构建脚本

* 更新libsysapi.a为libc.a
```

(下页继续)

(续上页)

```
    * 加上ferror fopen fclose

    * 更新移植的软件的构建脚本

    * 更改build_gcc_toolchain.sh中的-save参数名为-save-cache

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 61de2cdc3f29cdc6c441f128119e01e003e6f3ca
Author: login <longjin@ringotek.cn>
Date:   Tue Jan 3 23:09:25 2023 +0800

    新增rust版本的lockref (#135)

    * new:Rust封装cpu_relax(),通过pause指令，让cpu休息一会儿。降低空转功耗

    * new: Rust版本的lockref

    * Rust的RawSpinlock新增is_locked()和set_value()方法。

    * lockref文档

commit 2726f101b4cc787bbd36a69afffb0112f3a6567f
Author: login <longjin@ringotek.cn>
Date:   Tue Jan 3 21:01:56 2023 +0800

    删除无用的cfs.h (#136)

commit 587086d3f299f7394559d547c828191be20cfc11
Author: login <longjin@ringotek.cn>
Date:   Sun Jan 1 16:53:57 2023 +0800

    1、在文件系统目录下增加mod.rs 2、将VFS的路径改为vfs(#133)

    2、将VFS的路径改为vfs

commit 843e442971a47693f37a5f8d3452c383f7325359
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 31 18:43:05 2022 +0800

    修复init进程忘记设定fs gs寄存器的问题。 (#132)

commit 74bde36e014ff501241bf40dd83653db47a2c8e4
```

(下页继续)

(续上页)

```
Author: guanjinquan <1666320330@qq.com>
Date:   Sat Dec 31 17:35:39 2022 +0800

    Patch porting gcc v2 (#124)

    * 更改编译器的Include路径，使得include时不需要加`<libc/src/include/>`前缀

    * 修改include路径

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit d4f3de93a23e4bd4f000a3663768d47d094bf188
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 31 17:26:12 2022 +0800

    使用Rust重构CFS调度器 (#131)

    * 新建调度器的文件

    * 把softirq vector移动到c文件中（原来在.h)

    * 将进程切换方式改为“中断返回时切换”

    * new:使用rust重构CFS

    * 删除已经在smp中废弃的HPET中断转发函数

    * 代码格式化

    * 删除多余的dunce依赖

commit 156949680c83f2d7e3b21ed68b11698b88eaf396
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 31 13:47:49 2022 +0800

    ␣
↪bugfix:修复当使用sched()运行调度器，在切换进程的时候，由于不在中断上下文内，导致当前进程的上下文丢
↪(#130)

    ␣
↪bugfix:修复当使用sched()运行调度器，在切换进程的时候，由于不在中断上下文内，导致当前进程的上下文丢
    ␣
↪bugfix:修复切换进程的宏的汇编代码的损坏部分，未声明rax寄存器，从而导致的编译器未定义行为问题。
```

(下页继续)

(续上页)

```
commit 882f0b7e7498dbff8de527b2b9159b7f6e6359c9
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Wed Dec 28 19:35:17 2022 +0800

    修复内存bug与grub安装脚本的错误 (#129)

    * 修复内存bug与grub安装脚本的错误

    * 修改小bug

commit adc1846b06fb862caed049f435fc0061488a6ff9
Author: login <longjin@ringotek.cn>
Date:   Mon Dec 26 13:13:12 2022 +0800

    内核：在lib.
↪rs中，将arch模块的路径进行更改，使得其他模块使用arch的代码时，不需要指定arch::x86_
↪64 (#128)

commit ac643d420b22f9d454ecefccd51ed34a9664586b
Author: login <longjin@ringotek.cn>
Date:   Sun Dec 25 23:53:35 2022 +0800

    new:新增rust写的RawSpinlock (#127)

commit 998390210549b47e6bdcc3fdab49eff4086ad18b
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 24 23:30:26 2022 +0800

    新增signal文档 (#126)

    * 新增signal文档

commit a7f5ca7b67160557abf84a1169dd60093220aeb0
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Sat Dec 24 23:29:36 2022 +0800

    修复下载grub2.06时的提示错误 (#125)

    * 修复grub下载显示提示显示错误

commit 82762007da41148e1ed1df465211eb5c8ba2c15e
Author: login <longjin@ringotek.cn>
```

(下页继续)

(续上页)

```
Date:   Fri Dec 23 18:11:47 2022 +0800

    Update makefile.yml

commit b975025ec8854ca232152f4ee44cc2226891a34c
Author: login <longjin@ringotek.cn>
Date:   Fri Dec 23 11:45:19 2022 +0800

    Update makefile.yml

commit ad2bb74d949bfcb2935e43ac7b261d7ecce23389
Author: login <longjin@ringotek.cn>
Date:   Fri Dec 23 11:21:22 2022 +0800

    Update makefile.yml

commit 6b7776d189ab5f19fbab20d6c5c9ed3ab20c7ab6
Author: login <longjin@ringotek.cn>
Date:   Fri Dec 23 10:59:15 2022 +0800

    修正smp的makefile中没有替换AS的问题

commit beb12a188b6c6bc4196796ac2ae1ecd7d8ed8223
Author: login <longjin@ringotek.cn>
Date:   Fri Dec 23 10:57:39 2022 +0800

    Update makefile.yml

commit d65c527730e5c8a75f6dad0f996c093040699ee3
Author: login <longjin@ringotek.cn>
Date:   Thu Dec 22 22:58:28 2022 +0800

    Update makefile.yml (#121)

commit 5ed4cd460200cb19aae8c3c67dfd77e1e9f0e105
Author: guanjinquan <75822481+guanjinquan@users.noreply.github.com>
Date:   Thu Dec 22 21:09:12 2022 +0800

    Patch gcc toolchain (#111)

    * 添加了GCC_cross_compile——tool_chain

    * - 解决环境变量路径拼接时，多了`/`的问题
```

(下页继续)

(续上页)

```
    - apt安装时增加-y,不需用户确认

    * 解决添加环境变量的命令有误的问题

    * 修正编译错误时，还会执行下一步的问题

    * new: 编译完成后清理临时文件

    * 更新makefile

    * 调整：把grub安装在 $HOME/opt/dragonos-grub下

    * new: 新增dockerfile

    * 将镜像源换成中科大的（原因是清华的总是ban掉用于构建镜像的服务器的ip）

    * 修改为基于debian bullseye构建

    * 取消指定版本

    * 修复MBR磁盘镜像未设置启动标志的bug

    * 取消在docker中安装grub

    * 安装grub的过程改到客户机上进行

    * bootstrap.sh 添加--no-docker

    * 使用新版的docker编译镜像

    * 修补，添加了一些关于gcc的check

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit ba0d93d8b26034abc54bcaf3f0ff04863bbd076e
Author: Gou Ngai <94795048+AlbertSanoe@users.noreply.github.com>
Date:   Mon Dec 19 15:04:37 2022 +0800

    refactor rtc module in rust (#118)

    * 用rust重构rtc模块

    * refactor the rtc module by rust
```

(下页继续)

(续上页)

```
    * rtc-updated

    * rtc-updated-4

    * rtc

commit c588d6f77f4b38939701b946228218ea81a7c8dc
Author: login <longjin@ringotek.cn>
Date:   Mon Dec 19 15:03:44 2022 +0800

    Patch add abort func (#120)

    * 对于除了sigkill以外的信号，也加入队列

    * bugfix:libc中，注册信号处理函数时，总是注册sigkill的问题

    * 增加getpid系统调用

    * 增加了raise、kill、abort

commit 47f0d12a1f1a1aa11be8e751ecdbf76f0cb596d9
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Mon Dec 19 14:53:51 2022 +0800

    修复docker安装时异常退出的bug (#119)

    * 修复docker安装时异常退出的bug

    * 修复grub编译脚本的小bug

commit 978043e47d1143ca2d5cf22b20793f032e8eb5a5
Author: login <longjin@ringotek.cn>
Date:   Sun Dec 18 15:09:15 2022 +0800

    修复当系统中不存在dosfstools时，无法正确格式化磁盘镜像的问题 (#117)

    * 修复当系统中不存在dosfstools时，无法正确格式化磁盘镜像的问题

commit f9127772dc372a2e607388fdd6818d3f9c4c6d28
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Sat Dec 17 23:43:23 2022 +0800

    修复docker安装时异常退出的bug (#116)
```

21.8 V0.1.2

备注： 本文作者：龙进 longjin@RinGoTek.cn

2022 年 12 月 17 日

21.8.1 贡献者名单

DragonOS V0.1.2 版本由以下小伙伴贡献代码：

- 龙进 longjin@ringotek.cn
- 吴宇健 wuyujian@DragonOS.org
- Gou Ngai sujintao@DragonOS.org
- 黄厅 huangting@DragonOS.org
- 王文聪 1297389017@qq.com

21.8.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- David Wen
- Seele.Clover
- TerryLeeSCUT
- 悟
- slientbard

其中，非常感谢 **Seele.Clover** 给予 DragonOS 项目人民币 500 元的赞助与支持！我们对于每一笔赞助款项，将仔细登记，并确保其能被妥善的使用。

21.8.3 更新内容-内核

- 删除 rust_helloworld 文件 (#113)
- Signal: 允许用户注册信号处理函数，能够进入自定义的 handler。(#112)
  - 支持 kill 命令
  - 允许用户自定义信号处理函数
  - 新增 2 个系统调用：`SYS_SIGACTION`，`SYS_RT_SIGRETURN`

  - libc 增加 `signal()`,`sigaction()` 函数。
  - 暂时只支持旧版的 sighandler，即：只有 1 个参数的 `void handler(int signum)` 类型的信号处理函数。对于另一种信号处理函数 `void handler(int signum, siginfo_t *info, void* data)`，尚不支持传递第三个参数。

- 在内核代码中加入自定义的 stdint.h 文件 (#109)
- 调整编译 grub 的脚本的部分 (#108)
- 新增 32、64 位 uefi 启动 (#105)(#101)
- 使用编译安装的 grub-2.06，解决客户机上 grub 版本不对导致的编译无法运行的问题。
- 增加了 timekeeping 模块 (#106)
- bugfix: 修复 rtc 时钟对 BCD 码进行转换的时候，忘了处理 day 字段的问题 (#104)
- new: 开发过程文档（完成了一半）
- bootstrap.sh 解决下载 rust 慢的问题
- 更新“构建系统”文档
- procfs->status 增加显示 preempt 和虚拟运行时间 (#100)
- ffz 函数：获取 u64 中的第一个值为 0 的 bit (#100)
- 解决由于编译器优化导致 local_irq_restore 无法获取到正确的 rflags 的值的问题
- 使用 Rust 重构串口驱动 (#99)

21.8.4 更新内容-用户环境

- about app: 显示当前构建的 git commit sha1 以及构建时间 (#114)
- shell: 修复 shell 的 exec 命令对绝对路径的拼接错误问题 (#114)
- shell: exec 命令增加”&”后台运行选项 (#100)
- new: 测试 signal 用的 app
- 将 libc 目录进行调整，加入 cargo 作为 rust 的包管理器

21.8.5 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/fslongjin/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn

21.8.6 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT

21.8.7 当前版本的所有提交记录

```
commit 7a818da88a1c7a1760de7671141b0ce1ca4e3dde
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 17 17:49:12 2022 +0800

    Patch about auto gen version string (#114)

    * new: about app中，显示当前构建的git commit sha1以及构建时间

    * bugfix: 修复shell的exec命令对绝对路径的拼接错误问题

commit 83a7aaa46bbc411c43d4fc099c6c8884efbe4771
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 17 16:31:50 2022 +0800

    删除rust_helloworld文件 (#113)

commit 6efd4740336205c9bfdd8b164e667cee2f38781e
Author: login <longjin@ringotek.cn>
Date:   Sat Dec 17 16:27:50 2022 +0800

    允许用户自定义信号处理函数 (#112)
```

(下页继续)

(续上页)

```
    * new: 用户注册信号处理函数，能够进入自定义的handler

    * 修复忘了传信号的数字给用户的处理函数的bug

    * new:sigreturn

    * 删除注释

commit 0e0c187484281768391e131495f0655e40d70cf7
Author: login <longjin@ringotek.cn>
Date:   Fri Dec 16 16:20:09 2022 +0800

    在内核代码中加入自定义的stdint.h文件 (#109)

commit d02e6ea4112ad520aa4090ff73cdf592e14c0a82
Author: login <longjin@ringotek.cn>
Date:   Wed Dec 14 20:01:55 2022 +0800

    调整编译grub的脚本的部分 (#108)

    1、bugfix: 修复编译grub的脚本的部分错误
    2、将grub下载源替换为tuna
    3、优化写入磁盘镜像的脚本
    4、将bios文件夹改名为legacy

commit 38b341b8aa671f75ac26d05059aa2e9a09e653b7
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Wed Dec 14 16:58:49 2022 +0800

    新增32位uefi启动 (#105)

    * 新增32位uefi启动

    * 修复小bug

    * 增加grub本地编译安装

    * 增加本地grub编译安装脚本

    * 修正小错误

    * 修复空文件夹不上传的bug
```

(下页继续)

(续上页)

```
commit 01876902fbf6ed43992cc7d153bd8c505cb5224b
Author: Gou Ngai <94795048+AlbertSanoe@users.noreply.github.com>
Date:   Wed Dec 14 15:13:54 2022 +0800

    增加了timekeeping模块 (#106)

    * 增加了timekeeping模块

    * 格式化文档和细节更改

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 728aca308917a7d4d0ba10fe8174e9408d77a9a6
Author: login <longjin@ringotek.cn>
Date:   Sun Dec 11 22:59:47 2022 +0800

    bugfix: 修复rtc时钟对BCD码进行转换的时候，忘了处理day字段的问题 (#104)

commit 237e95c6ddce72d72ae7fedfeca412fab82b3622
Author: wwc-15172310230 <78997674+wwc-15172310230@users.noreply.github.com>
Date:   Sun Dec 11 22:22:10 2022 +0800

    调整user下libs的libc目录结构 (#103)

    * 调整user下libs的libc目录结构

    * 修正.gitignore文件的问题

    * 修复无法编译的问题

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 2291ffdece1dc5a703602f79f74df8a4854d215b
Author: login <longjin@ringotek.cn>
Date:   Sun Dec 11 20:09:58 2022 +0800

    文档更新 (#102)

    * new: 开发过程文档（完成了一半）

    * bootstrap.sh解决下载rust慢的问题
```

(下页继续)

(续上页)

```
    * 更新“构建系统”文档

commit 7f439c5ddbd2ecffc112149d16983975f523052c
Author: YJwu2023 <119829947+YJwu2023@users.noreply.github.com>
Date:   Fri Dec 9 16:08:54 2022 +0800

    增加uefi启动 (#101)

    * 增加uefi启动

    * 修改脚本

    * uefi修改

    * 删除错误的注释

    * 修正写入磁盘镜像的脚本

    * 修改X86_64为x86_64

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 1a2eaa402f05f82aaeebe1e03824534a0a425d4d
Author: login <longjin@ringotek.cn>
Date:   Thu Dec 8 22:59:51 2022 +0800

    signal的处理(kill命令)以及一些其他的改进 (#100)

    * 将entry.S中冗余的ret_from_syscall代码删除，改为jmp Restore_all

    * new: 增加判断pt_regs是否来自用户态的函数

    * new: rust的cli和sti封装

    * 将原有的判断pt_regs是否来自用户态的代码，统一改为调用user_mode函数

    * ffz函数：获取u64中的第一个值为0的bit

    * spinlock增加 spinlock irq spin_unlock_irq

    * 临时解决显示刷新线程迟迟不运行的问题

    * 更改ffi_convert的生命周期标签
```

(下页继续)

(续上页)

```
    * new: 测试signal用的app

    * 解决由于编译器优化导致local_irq_restore无法获取到正确的rflags的值的问题

    * new: exec命令增加"&"后台运行选项

    * procfs->status增加显示preempt和虚拟运行时间

    * 更改引用计数的FFIBind2Rust trait中的生命周期标签

    * new: signal处理(kill)

    * 更正在review中发现的一些细节问题

commit f8b55f6d3fcbf152a1cb6d6fc722bf1607418b28
Author: TingHuang <92705854+TingSHub@users.noreply.github.com>
Date:   Tue Dec 6 22:15:03 2022 +0800

    Patch uart (#99)

    * 添加UART驱动相关文件

    * 添加驱动核心文件，将rust编写的驱动代码加入Package中

    * 添加glib.h文件生成rust代码，添加uart驱动代码

    * 添加串口发送及接收相关代码

    * 添加字符串发送函数,未实现具体功能

    * 为调用uart驱动的代码添加rust接口

    * 添加字符串发送函数，修改C语言调用接口

    * 添加rust串口驱动

    * 添加uart.h头文件，将串口端口类型改为enum

    * 添加注释，规范代码

commit 036acc52ce9d0fb9e7d92768ff74939a29c07f32
Author: login <longjin@ringotek.cn>
```

(下页继续)

(续上页)

```
Date:   Tue Nov 29 21:46:13 2022 +0800

    将entry.S中冗余的ret_from_syscall代码删除，改为jmp Restore_all (#98)

    * 将entry.S中冗余的ret_from_syscall代码删除，改为jmp Restore_all
```

21.9 V0.1.1

备注： 本文作者：龙进 longjin@RinGoTek.cn

2022 年 11 月 27 日

21.9.1 贡献者名单

DragonOS V0.1.1 版本由以下小伙伴贡献代码：

- 龙进 longjin@RinGoTek.cn
- 周于🄵 zhouyuzhe@DragonOS.org

21.9.2 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- David Wen
- TerryLeeSCUT
- 悟
- slientbard

其中，非常感谢 **David Wen** 给予 DragonOS 项目人民币 1000 元的赞助与支持！我们对于每一笔赞助款项，将仔细登记，并确保其能被妥善的使用。

21.9.3 更新内容-内核

- 新增 rust ffi (#77)
- port kmalloc and printk to rust
- rust 下的 kdebug kinfo kwarn kBUG kerror 宏
- bugfix: 修复进程 pcb 被回收时，未将其从链表中删除的问题
- 目录结构优化：移动 asm.h 和 cmpxchg.h
- signal 的发送
- procfs：查看进程的 status
- 解决第一次编译时磁盘镜像权限错误的问题
- 将 fork 相关代码移动到 fork.c

21.9.4 更新内容-用户环境

- shell：增加 kill 命令，可向目标进程发送信号。但由于仍未完善 signal 机制，因此目标进程暂时不能响应这个信号。

21.9.5 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/fslongjin/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn

21.9.6 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT

21.9.7 当前版本的所有提交记录

```
commit d65ade9c5909076747bd00966a398fe27fbd290d
Author: DaJiYuQia <88259094+DaJiYuQia@users.noreply.github.com>
Date:   Sun Nov 27 14:21:31 2022 +0800

    Patch procf (#95)

    * debug color problem

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit 6cb769c423b09e88fea1763210200a716477be0a
Author: login <longjin@ringotek.cn>
Date:   Sun Nov 27 14:17:36 2022 +0800

    将include目录下的rust代码转移到他们应当属于的模块中 (#96)

    * 将include目录下的rust代码转移到他们应当属于的模块下。

commit 27a97abd2474b03ad09b562e5ed11e1fdae8eb32
Author: DaJiYuQia <88259094+DaJiYuQia@users.noreply.github.com>
Date:   Sat Nov 26 17:34:00 2022 +0800

    Patch procf (#90)

    * 1234

    * 123

    * 合并master

    * procfs

    * 1

    * procfs展示进程基本信息

    * modified code

    * 恢复权限

    * 恢复权限
```

(下页继续)

(续上页)

```
     #恢复权限

    * modify permission

    * 删除run.sh

    * 解决第一次编译时磁盘镜像权限错误的问题

    * 恢复.vscode/c_cpp_properties.json

    * 删除process.c中错误的do_fork

    * remake procfs

    * 修改一些变量名

    * 修改类型

    * modified

    * data_puts缓冲区溢出后return

    Co-authored-by: longjin <longjin@RinGoTek.cn>

commit ad23fcddf893d7f92d2bf3efdb66e969416d2852
Author: login <longjin@ringotek.cn>
Date:   Wed Nov 23 21:34:35 2022 +0800

    bugfix: 修复进程退出时未释放signal和sighand && 增加赞赏者名单：David Wen (#93)

    * bugfix: 修复进程退出时未释放signal和sighand的bug

    * 增加赞赏者名单：David Wen

commit 0274cd6eeec01885232e7418a501857cb76da69e
Author: login <longjin@ringotek.cn>
Date:   Wed Nov 23 20:43:18 2022 +0800

    修正drop signal结构体的box对象的的问题 (#92)

    * fix: exit signal and exit sighand

commit c8025a88798dc57ecc5d7f20ad69de695445638f
```

(下页继续)

(续上页)

```
Author: login <longjin@ringotek.cn>
Date:   Wed Nov 23 20:18:22 2022 +0800

    new:在fork时拷贝signal和sighand (#91)

    * refcount初始化

    * new: 实现copy_sighand
    del: 删除sighand_struct的wqh, 待将来有需要时, 替换成rust版本的

    * new: 拷贝signal
    bugfix: 解决拷贝sighand时的uaf问题

commit 66f67c6a95b8aad85cfd2146a86e5e3e6a3568e7
Author: login <longjin@ringotek.cn>
Date:   Wed Nov 23 11:38:20 2022 +0800

    signal的发送（暂时父子进程之间共享信号及相应的结构体） (#89)

    * 解决由于spinlock.h中包含preempt_enable()带来的循环include问题

    * new: 初步实现signal的数据结构

    * new:signal相关数据结构

    * fix: 解决bindings.rs报一堆警告的问题

    * new: rust下的kdebug kinfo kwarn kBUG kerror宏

    * 移动asm.h和cmpxchg.h

    * new: signal的发送（暂时只支持父子进程共享信号及处理函数）

commit 3d729e2069e01ee07525ff83167566dac5322a40
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 18 17:59:33 2022 +0800

    bugfix: 修复进程pcb被回收时，未将其从链表中删除的问题 (#87)

    * bugfix: 修复进程pcb被回收时，未将其从链表中删除的问题
    new: pcb相关api文档

    * 将文档加入目录
```

(下页继续)

(续上页)

```
commit 0bfe94f46be9bdde1ade81a20e803aa2aafd2964
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 18 16:32:15 2022 +0800

    new: rust下的kdebug kinfo kwarn kBUG kerror宏 (#86)

    * new: rust下的kdebug kinfo kwarn kBUG kerror宏

commit c6174797dcf3427f38bfa0f4bd3e039c319f7c5b
Author: login <longjin@ringotek.cn>
Date:   Thu Nov 17 20:29:29 2022 +0800

    fix: 解决bindings.rs报了很多警告的问题 (#85)

    * fix: 解决bindings.rs报一堆警告的问题

commit cffd7144fbed84f9775e89d7b99602c6ccc5a510
Author: login <longjin@ringotek.cn>
Date:   Wed Nov 16 15:18:03 2022 +0800

    signal相关数据结构&代码结构优化 (#84)

    * 解决由于spinlock.h中包含preempt_enable()带来的循环include问题

    * new: 初步实现signal的数据结构

commit fb6c29d01d4cf92368efec08c01e419c2a941f7d
Author: login <longjin@ringotek.cn>
Date:   Sun Nov 13 16:43:58 2022 +0800

    port kmalloc and printk to rust (#83)

    * 暂时移除cbindgen

    * 将lib文件夹更名为libs文件夹（解决rust的冲突）

    * 实现了全局的allocator

    * 实现了printk宏

    * new: 完善了printk的颜色
```

(下页继续)

(续上页)

```
commit 82d2e446a401e7eee57a847f48a6d162931170c3
Author: login <longjin@ringotek.cn>
Date:   Sat Nov 12 15:25:54 2022 +0800

    new: 暂时移除cbindgen (#82)

commit 2aaf7808efe44ecfaadd51ae4f8892e667108578
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 22:21:44 2022 +0800

    在内核中引入cbindgen, 生成rust-C的FFI (#81)


    * 解决codeql失败问题

    * new: 为内核引入cbindgen

commit 2813126e3190c9b3c1a836a647b259a7adbe0cf3
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:35:37 2022 +0800

    新增rust ffi (#77)

    * 引入cargo

    * 取消对Cargo.lock的跟踪

    * 解决vscode报错问题

    * new: rust的代码能够调用c语言的printk_color

    * 1、将原本run.sh的工作拆解，变为几个不同的make命令
    2、在docker镜像中编译rust

    * 更改workflow

    * update workflow

    * new: 解决workflow无法通过编译的问题

commit 5e023cf7911333eb05bfe65704dce4b01fa4d0a7
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:21:45 2022 +0800
```

(下页继续)

(续上页)

```
    Update makefile.yml

commit e44795008f7e34d2068cf28dcedbcb91f5ccd66b
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:18:13 2022 +0800

    Update makefile.yml (#80)

commit ec5fb84b61c313824cc2199ab64e3af4b7e5f895
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:08:09 2022 +0800

    Update makefile.yml

commit 6d9dff5f1ff347ea780a0249e54eef356cdcaaea
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:07:48 2022 +0800

    Revert "Update makefile.yml (#78)" (#79)

    This reverts commit badc7d238f2341e844a90be3e357e5dd77a447fc.

commit badc7d238f2341e844a90be3e357e5dd77a447fc
Author: login <longjin@ringotek.cn>
Date:   Fri Nov 11 15:05:52 2022 +0800

    Update makefile.yml (#78)
```

21.10 V0.1.0

备注： 本文作者：龙进 longjin@RinGoTek.cn

2022 年 11 月 6 日

21.10.1 前言

DragonOS 从 2022 年 1 月 15 日开始开发，到如今已经经历了将近 300 天。在这么多个日夜里，已经数不清到底花了多少时间在 DragonOS 的开发之中，我基本上把所有的空闲时间都给了 DragonOS，保守估计总工时已经在 1000 小时以上。能够发布第一个版本，我感到十分有成就感。

在 2022 年 7 月以来，陆陆续续的，有来自 6 所高校或企业的小伙伴/大佬加入了 DragonOS 的开发。我当时非常的欣喜，我想，也许在大家的一同努力下，我们能创造出一个真正具有实用性的操作系统呢！我们累计召开了 14 次交流讨论会。我相信，在大家的共同努力下，将来，我们一定能创造出真正独立自主的、开放的、面向服务器领域应用的开源操作系统，并在生产环境中得到应用。

尽管 DragonOS 目前只是一个玩具水平的操作系统，只是“比本科生毕业设计难度略高的”操作系统。但是，请不要小看它，它的内在的架构设计，瞄准了 Linux5.18 及以后的发行版，虽尚未能达到 Linux 的水平，但我们正努力追赶。得益于 Linux 的相关资料，DragonOS 在架构设计之时，学习了 Linux 的很多设计思想，相关组件都尽量考虑了可扩展性与可移植性。

千里之行，始于足下。DragonOS V0.1.0 版本的发布，是一个全新的开始。**希望在未来的十年里，我们能与众多伙伴们一同努力，在 2032 年，将 DragonOS 建设成为具有实用意义的，能够在服务器领域取得广泛应用的开源操作系统！**

百舸争流，奋楫者先；中流击水，勇进者胜。我相信，在接下来的时间里，在社区开发者们的不断努力下，我们的目标，终将成为现实！

21.10.2 特别鸣谢

在 DragonOS V0.1.0 版本的发布之际，我想对我的老师、前辈以及学校表示衷心的感谢！

- **佛山市南海区大沥镇中心小学姚志城老师**：您是带领我接触计算机，学会编程的领路人。十年前，与您交谈时，您说过：“我们国家目前还没有自主的、成熟的操作系统”。这句话，为我的梦想埋下了种子。您培养了我对计算机的热爱，因此我选择了软件工程这个专业。感谢当年您的教导，师恩难忘！
- **佛山市南海区石门实验学校**：在石实就读的三年里，非常感谢石实的“扬长教育”理念，在老师们的培养下，让我充分发挥了自己的个性和特长，也取得了不错的成绩。在石实的三年里，我学会了 C++、Java 以及简单的算法，也自己开发了几个安卓 app，积累了将近 6 千行的代码量。
- **佛山市南海区石门中学**：“任重道远，毋忘奋斗”是石中的校训，我想，这句校训，也应当成为我们每个新时代青年人的座右铭。在石门中学的三年，家国情怀教育对我产生了很大的影响。我想，我们作为新时代的青年，更应当肩负起时代的重任，奋勇拼搏，为祖国的发展，为民族的自强，为人类的未来，努力奋斗！
- **华南理工大学**：“博学慎思，明辨笃行”，在华工，我得到了进一步的学习与发展。开拓了自己的视野，学会了跟很多人打交道。并且，在软件学院，我遇到了一群认真负责的老师。非常感谢学院对我的支持，支持我们成立项目组。我相信，在学院的支持下，能让 DragonOS 取得更好的发展，走的更远！
- **华南理工大学软件学院王国华老师**：王老师是我《操作系统》课程的老师，在她的指导下，我对操作系统的原理有了更深的理解，并参加了“泛珠三角 + 大学生计算机作品赛“，在 2022 年 6 月的广东省选拔赛中，DragonOS 取得了一等奖、最佳创新奖的好成绩。

- **华南理工大学软件学院汤峰老师**：汤老师是我们在校内的项目组的指导老师。在她的悉心指导下，我们将不断前行，保持我们前进的方向，持续构建开源社区。我由衷地感谢汤老师的指导！
- **Yaotian Feng**：在 Bilibili 上认识了这位非常厉害的老哥，解答了我的很多问题，很多次在我毫无头绪的 debug 了几天之后，几句话点醒了我，让我找到解决问题的路径。并且，他也跟我分享了容易踩坑的地方，让我在将要踩坑的时候能够有个心理预期，不至于那么难受哈哈哈哈。

21.10.3 贡献者名单

DragonOS V0.1.0 版本的发布，离不开以下小伙伴们的共同努力：

- 龙进 longjin@RinGoTek.cn
- zzy666-hw zzy666@mail.ustc.edu.cn
- 关锦权 guanjinquan@DragonOS.org
- 周于🄵 zhouyuzhe@DragonOS.org
- kkkkkong kongweichao@DragonOS.org
- houmkh jiaying.hou@qq.com
- wang904 1234366@qq.com
- Liric Mechan i@liric.cn
- Mustang handsomepd@qq.com
- Eugene caima12138@foxmail.com
- kun 1582068144@qq.com
- zhujikuan 1335289286@qq.com
- Alloc Alice 1548742234@qq.com

21.10.4 赞助者名单

感谢以下同学的赞赏，我们将不断努力！

- TerryLeeSCUT
- 悟
- slientbard

21.10.5 内核

遵循的一些标准规范

- 启动引导：Multiboot2
- 系统接口：posix 2008

硬件架构

- 目前支持在 x86-64 架构的处理器上运行

Bootloader

- 使用 Grub 2.06 作为 bootloader

内存管理

- 实现了基于 bitmap 的页分配器
- 实现了 slab 分配器，用来分配小块的、具有对齐要求的内存
- 抽象出 VMA（虚拟内存区域）
- 实现 VMA 反向映射机制
- 实现 MMIO 地址空间自动映射机制

多核

- 支持多核引导。也就是说，在 DragonOS 启动后，将会启动 AP 处理器。但是，为了简化其他内核模块的实现，目前 AP 处理器上，暂时没有任务在运行。
- 粗略实现了 IPI（处理器核间通信）框架

进程管理

- 支持进程的创建、回收
- 内核线程
- kthread 机制
- 用户态、内核态进程/线程的 fork/vfork（注意，用户态的 fork 和内核态的有一定的区别，内核态的 fork 更复杂）
- exec 让进程去执行一个新的可执行文件

- 进程的定时睡眠（sleep)（支持 spin/rdtsc 高精度睡眠、支持上下文切换方式的睡眠）

同步原语

- spinlock 自旋锁
- mutex 互斥量
- atomic 原子变量
- wait_queue 等待队列
- semaphore 信号量

调度相关

- CFS 调度器
- 单核调度（暂时不支持多核负载均衡）
- completion “完成”机制，让一个进程能等待某个任务的完成。

IPC 进程间通信

- 匿名管道

文件系统

- VFS 虚拟文件系统的基本功能
- FAT32 文件系统（尚不支持删除文件夹）
- devfs 设备文件系统。目前只将键盘文件注册到其中。
- rootfs 根文件系统，在真实的磁盘文件系统被挂载前，为其他的伪文件系统提供支持。
- 挂载点抽象。目前实现了文件系统的挂载，使用类似于栈的方式管理所有的挂载点。（将来需要优化这部分）

异常及中断处理

- 处理器异常的捕获
- 对 APIC 的支持
- softirq 软中断机制
- 能够对内核栈进行 traceback

内核数据结构

- 普通的二叉树
- kfifo 先进先出缓冲区
- 循环链表
- IDR 映射数据结构
- IDA ID 分配数据组件

屏幕显示

- VESA VBE 显示芯片驱动
- 实现了屏幕管理器，支持多个显示框架注册到屏幕管理器中。
- 实现了 TextUI 文本界面框架，能够渲染文本到屏幕上。并且预留了上下滚动翻页、多显示窗口的支持。
- printk

内核实用库

- 字符串操作库
- ELF 可执行文件支持组件
- 基础数学库
- CRC 函数库

软件移植

- 移植了 LZ4 压缩库（V1.9.3），为将来实现页面压缩机制打下基础。

内核测试

- ktest 单元测试框架
- 支持使用串口（COM1）输出屏幕内容到文件之中。

驱动程序支持

- IDE 硬盘
- AHCI 硬盘（SATA Native)
- ACPI 高级电源配置模块
- PCI 总线驱动
- XHCI 主机控制器驱动（usb3.0)
- ps/2 键盘
- ps/2 鼠标
- HPET 高精度定时器
- RTC 时钟
- local APIC 定时器
- UART 串口（支持 RS-232)
- VBE 显示
- 虚拟 tty 设备

系统调用

DragonOS 目前一共有 22 个有效的系统调用。

- SYS_PUT_STRING 往屏幕上打印字符
- SYS_OPEN 打开文件
- SYS_CLOSE 关闭文件
- SYS_READ 读取文件
- SYS_WRITE 写入文件
- SYS_LSEEK 调整文件指针
- SYS_FORK fork 系统调用
- SYS_VFORK vfork 系统调用
- SYS_BRK 调整堆大小为指定值
- SYS_SBRK 调整堆大小为相对值
- SYS_REBOOT 重启（将来 sysfs 完善后，将删除这个系统调用，请勿过度依赖这个系统调用）
- SYS_CHDIR 切换进程的工作目录
- SYS_GET_DENTS 获取目录中的目录项的元数据

- SYS_EXECVE 让当前进程执行新的程序文件
- SYS_WAIT4 等待进程退出
- SYS_EXIT 退出当前进程
- SYS_MKDIR 创建文件夹
- SYS_NANOSLEEP 纳秒级睡眠（最长 1 秒）在小于 500ns 时，能够进行高精度睡眠
- SYS_CLOCK 获取当前 cpu 时间
- SYS_PIPE 创建管道
- SYS_MSTAT 获取系统当前的内存状态信息
- SYS_UNLINK_AT 删除文件夹或删除文件链接

Rust 支持

- 实现了一个简单的 rust 语言的 hello world，计划在接下来的版本中，逐步转向使用 rust 进行开发。

21.10.6 用户环境

LibC

LibC 是应用程序与操作系统交互的纽带。DragonOS 的 LibC 实现了一些简单的功能。

- malloc 堆内存分配器
- 基础数学库
- 简单的几个与文件相关的函数
- pipe
- fork/vfork
- clock
- sleep
- printf

Shell 命令行程序

- 基于简单的字符串匹配的解析（不是通过编译课程学的的那一套东西做的，因此比较简单，粗暴）
- 支持的命令：ls,cd,mkdir,exec,about,rmdir,rm,cat,touch,reboot

用户态驱动程序

- 用户态键盘驱动程序

21.10.7 源码、发布版镜像下载

您可以通过以下方式获得源代码:

通过 Git 获取

- 您可以访问https://github.com/fslongjin/DragonOS/releases下载发布版的代码，以及编译好的，可运行的磁盘镜像。
- 我们在 gitee 上也有镜像仓库可供下载：https://gitee.com/DragonOS/DragonOS

通过 DragonOS 软件镜像站获取

为解决国内访问 GitHub 慢、不稳定的问题，同时为了方便开发者们下载 DragonOS 的每个版本的代码，我们特意搭建了镜像站，您可以通过以下地址访问镜像站：

您可以通过镜像站获取到 DragonOS 的代码压缩包，以及编译好的可运行的磁盘镜像。

- https://mirrors.DragonOS.org
- https://mirrors.DragonOS.org.cn

21.10.8 开放源代码声明

备注： 为促进 DragonOS 项目的健康发展，DragonOS 以 GPLv2 开源协议进行发布。所有能获得到 DragonOS 源代码以及相应的软件制品（包括但不限于二进制副本、文档）的人，都能享有我们通过 GPLv2 协议授予您的权利，同时您也需要遵守协议中规定的义务。

这是一个相当严格的，保护开源软件健康发展，不被侵占的协议。

对于大部分的善意的人们而言，您不会违反我们的开源协议。

我们鼓励 DragonOS 的自由传播、推广，但是请确保所有行为没有侵犯他人的合法权益，也没有违反 GPLv2 协议。

请特别注意，对于违反开源协议的，尤其是**商业闭源使用以及任何剽窃、学术不端行为将会受到严肃的追责**。（这是最容易违反我们的开源协议的场景）。

并且，请注意，按照 GPLv2 协议的要求，基于 DragonOS 修改或二次开发的软件，必须同样采用 GPLv2 协议开源，并标明其基于 DragonOS 进行了修改。亦需保证这些修改版本的用户能方便的获取到 DragonOS 的原始版本。

您必须使得 DragonOS 的开发者们，能够以同样的方式，从公开渠道获取到您二次开发的版本的源代码，否则您将违反 GPLv2 协议。

关于协议详细内容，还敬请您请阅读项目根目录下的 **LICENSE** 文件。请注意，按照 GPLv2 协议的要求，**只有英文原版才具有法律效力**。任何翻译版本都仅供参考。

开源软件使用情况

DragonOS 在开发的过程中，参考了一些开源项目的设计，或者引入了他们的部分代码，亦或是受到了他们的启发。现将他们列在下面。我们对这些开源项目的贡献者们致以最衷心的感谢！

格式：< 项目名 > - < 链接 > - < 开源协议 >

- Linux - https://git.kernel.org/ - GPLv2
- skiftOS - https://github.com/skiftOS/skift - MIT
- FYSOS - https://github.com/fysnet/FYSOS - FYSOS’ License
- LemonOS - https://github.com/LemonOSProject/LemonOS.git - BSD 2-Clause License
- LZ4 - https://github.com/lz4/lz4 - BSD 2-Clause license
- SerenityOS - https://github.com/SerenityOS/serenity.git - BSD 2-Clause license
- MINE - 《一个 64 位操作系统的设计与实现》田宇; 人民邮电出版社
- chcore - 《现代操作系统：设计与实现》陈海波，夏虞斌; 机械工业出版社
- SimpleKernel - https://github.com/Simple-XX/SimpleKernel - MIT

CHAPTER 22

Indices and tables

- genindex
- modindex
- search