HANA OS

HANAOS 是一个支持 riscv64 与 loongarch64 两种架构的宏内核操作系统。

整个启动过程，内核做了以下事情：

1. `U-Boot` 完成硬件初始化和加载内核后，将控制权转交给内核入口点（entrypoint），然后从入口点开始执行代码，最终跳转到 `main()` 函数
2. `main()` 函数承担了各个模块的初始化操作：
   - 串口初始化，启用打印输出功能；
   - 物理内存分配器，与虚拟内存管理的初始化；
   - 中断异常处理初始化；
   - `init` 进程初始化；块设备初始化；
   - 虚拟文件系统初始化。
3. 最后调用 `scheduler()` 调度器，加载 init 进程，开启下一个阶段的运行

HANAOS 基于类 Unix 的进程模型，支持多任务并发执行，并提供了丰富的系统调用接口供用户程序调用。

- 进程状态机设计如下：

```
INIT → RUNNABLE ↔ RUNNING → ZOMBIE
           ↓
       SLEEPING → (wakeup) → RUNNABLE
```

- 所有的系统调用功能参考 POSIX 接口实现

  以 clone 为例，接口如下：

```
SYSCALL_DEFINE5(clone, int,
unsigned long, flags, void*, stack, void*, ptid, void*, tls, void*, ctid)
```

HANAOS 的调度机制是由 `scheduler()`，`sched()`，`yield()` 共同完成的。

1. 在 `scheduler()` 内部的循环中，调度器会不停地遍历 proc_list，找到一个处于 RUNNABLE 状态的进程，接着调用 `swtch` 调度它。
2. `swtch()` 会将当前调用者需要保存的寄存器存放在 `mycpu()` 的结构体中，并将调度的进程的上下文加载，这样，就成功地切换到了选择的进程。
3. 在这个进程经过 `yield -> sched ->swtch` 的过程，将当前的保存寄存器放在自身的进程结构体的 `context` 内，并将 `mycpu()` 里面的寄存器加载出来，就这样回到了 `scheduler`。

对于 scheduler 来说，这个过程就像是在中途调用了一个名为 swtch 的普通函数一样；同理，对于调用 yield 最后 swtch 的进程来说也是这样。

HANAOS 的内核空间布局设计如下：

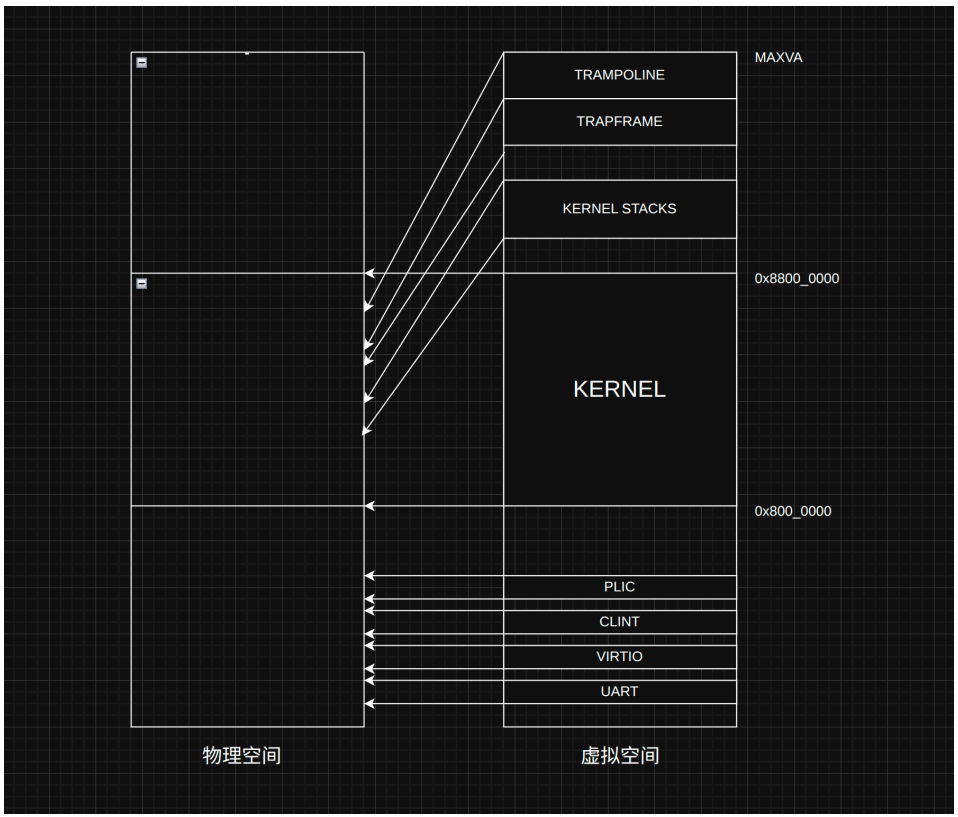


Figure 1: RISC-V

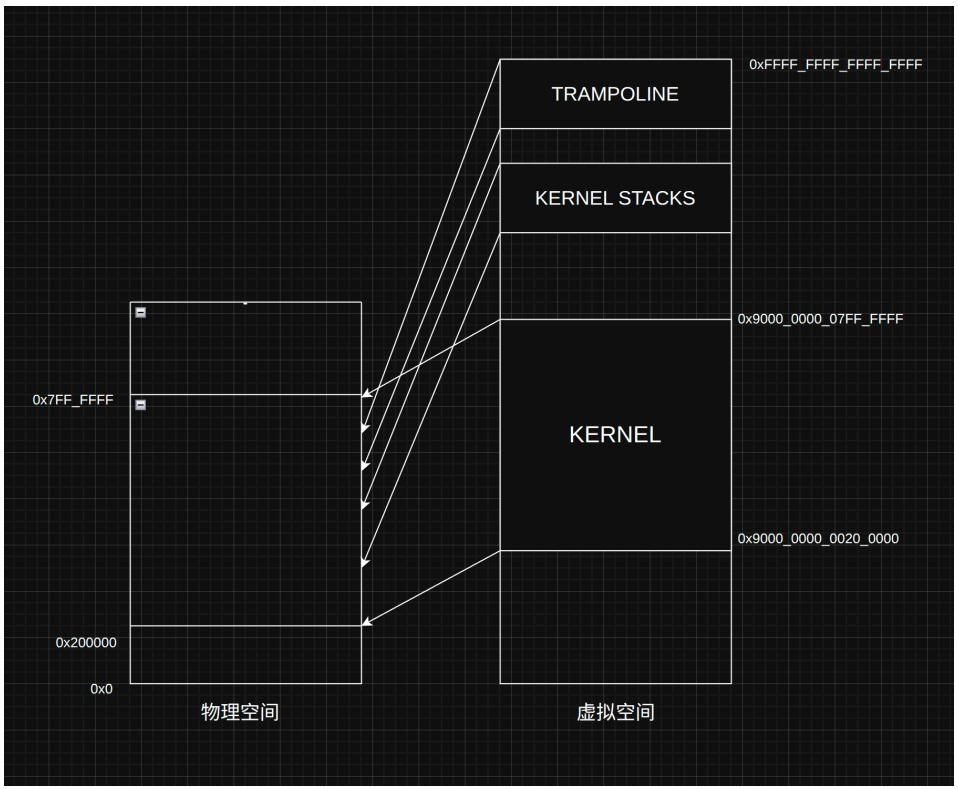


Figure 2: LoongArch

HANAOS 的用户空间布局设计如右图

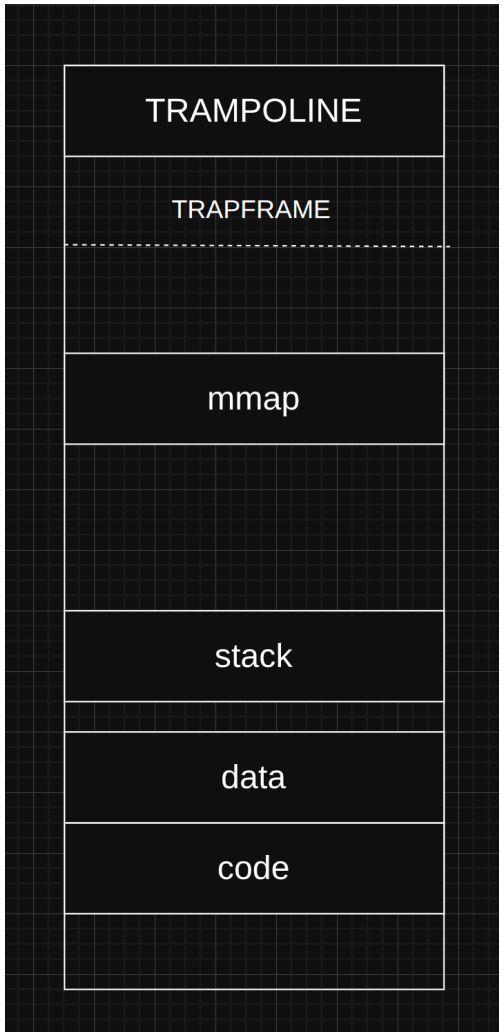


Figure 3: User-Space

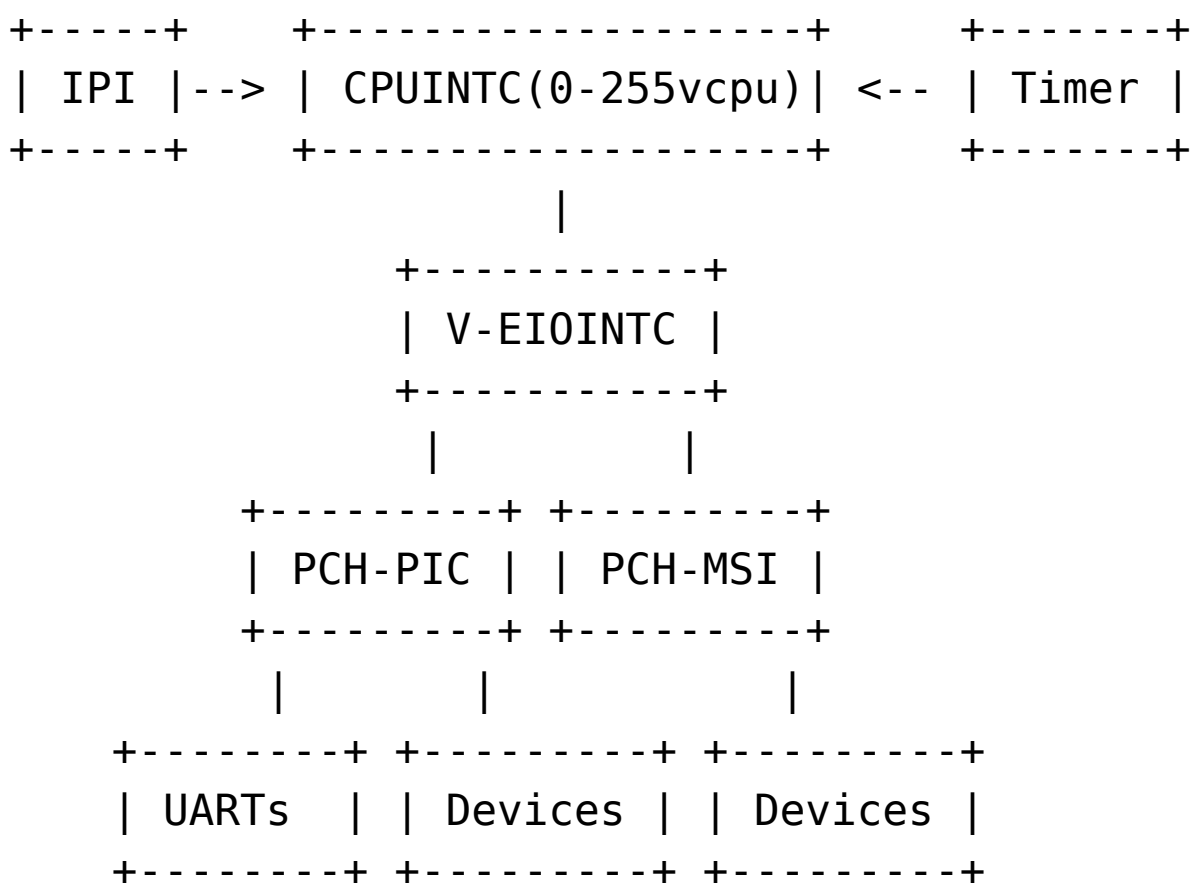
使用了伙伴系统和 *SLAB* 分配器。

- 如果 sz $>$ 3904，那么会使用伙伴系统分配，伙伴系统只会分配 4KB, 8KB, 16KB, 32KB, 64KB ... 4MB 大小的连续物理内存；
- 反之，则会使用 SLAB 分配器。SLAB 分配器会向伙伴系统申请 4KB 大小的块，称为 slab，又将 slab 分成数个小块，称为 object。每个 object 大小为 64B，作为最小分配单元。

HANAOS 实现了通用的中断子系统，并针对两种架构下的中断控制方式进行了驱动的适配。

- 对于 RISC-V，子系统调用 PLIC 控制器驱动来控制中断；
- 对于龙芯来说，其具有更复杂的中断路由，子系统支持使用 pch-pic 控制器的传统中断和使用 pch-msi 控制器的 msi 中断，二者都通过 eiointc 被通用子系统使用。

龙芯的外部中断遵循虚拟扩展 IRQ 模型，如下图：



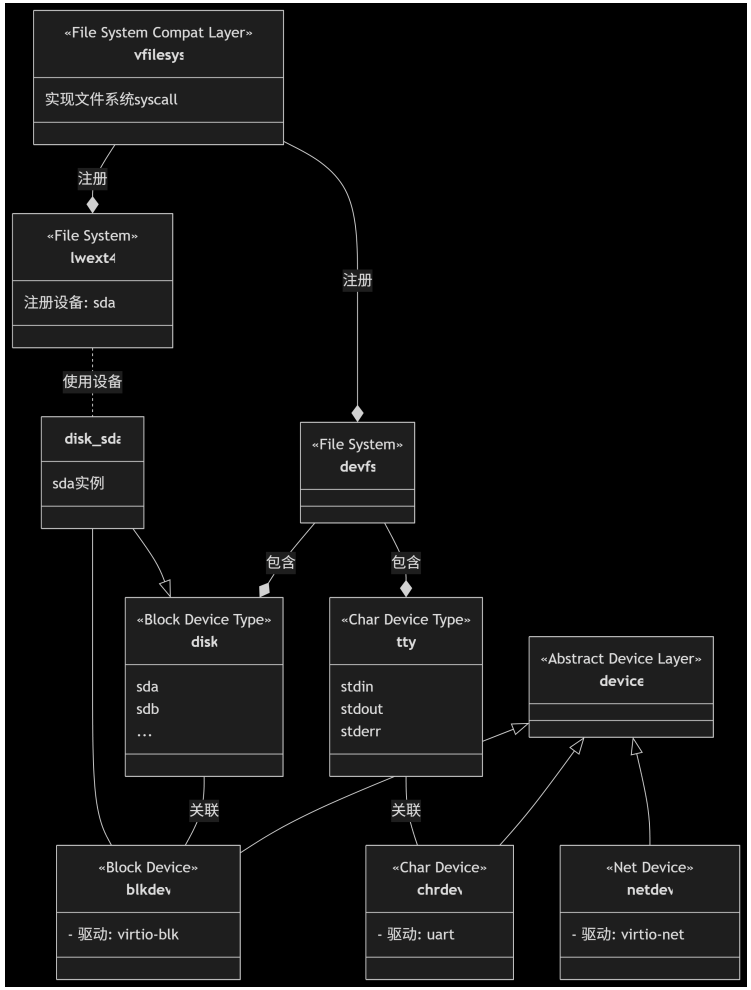


Figure 4: IO-stack

HANAOS 的 *I/O* 栈整体架构 设计如左图

HANAOS 采用 VIRTIO 作为底层 I/O 协议和网络栈物理层协议，实现了遵守 VIRTIO 0.95 版本。

- 在 RISCV 架构下基于 MMIO 总线。
- 在 Loongarch 架构下基于 PCI 总线并支持 MSI-X 中断，包含 Virtio-blk 设备和 Virtio-net 设备驱动。

HANAOS 还在龙芯下实现了 PCI 驱动，完成了 IO 和 MEM 两种 BAR 空间的配置和映射、PCI 传统中断的配置和中断号的映射、MSI-X 中断的配置等核心功能和模块

HANAOS 为各种各样的设备设计了统一的设备抽象 `struct device` 作为设备管理子系统的核心对象。

围绕这个类，设备管理子系统将各个种类的设备保存在一个链表中，提供统一的注册和初始化操作，为驱动提供设备的注册接口，管理中断操作。

HANAOS 支持如下的设备类：

- `struct device`: 通用的设备抽象。
- `struct blkdev`: 块设备类。随机读写设备的抽象，也就是磁盘设备驱动的抽象；
- `struct chrdev`: 单字符读写设备。如串口、鼠标、键盘等；
- `struct netdev`: 网络设备。物理层网卡设备的抽象；

HANAOS 实现了轻量级的，基于路径字符串的文件系统兼容层，该兼容层提供了文件系统的注册接口以及文件相关操作的系统调用。

兼容层的核心功能如下：

- 文件抽象结构：文件打开时保存的文件结构 `struct file` 和元数据信息 `struct inode`。
- 注册的文件系统接口：提供 `struct file_operations` 和 `struct fs_operations` 供文件系统注册文件和文件系统相关的操作
- 挂载点和文件系统抽象
- 设备管理系统 devfs：以文件的形式访问 I/O 设备，包含了 tty 和 disk 两种设备类型，自定义缓冲区和设备文件操作
- 管道的支持：通过注册文件操作接口，像读写文件一样读写管道

HANAOS 适配了 lwext4 库，实现了 lwext4 官方示例中的几个标准接口：

```
int blockdev_open(struct ext4_blockdev *bdev);
int blockdev_bread(struct ext4_blockdev *bdev, void *buf, uint64 blk_id,
       uint32 blk_cnt);
int blockdev_bwrite(struct ext4_blockdev *bdev, const void *buf,
        uint64 blk_id, uint32 blk_cnt);
int blockdev_close(struct ext4_blockdev *bdev);
int blockdev_lock(struct ext4_blockdev *bdev);
int blockdev_unlock(struct ext4_blockdev *bdev);
```