# Arm64操作系统实现原理札记

# 前言

作者目前就职于国际知名通信厂商，从事ARM嵌入式领域约10年。硬件平台从C51、MSP430、STM8、STM32到ARM Cortex-M0,M3M4，再到Cortex-A53,A57的功能开发。也曾在Altera的FPGA上开发过图像缓存帧调度算法，基于X86开发过视频图像驱动芯片。软件系统从裸机代码到ucosii，再到zephyer，到vxworks，到qnx，到linux，到fuchsia。最终深耕在ARM v8的架构的芯片，微内核操作系统领域、嵌入式虚拟化领域的软件开发。

写这本书的目的一是给自己这么多年的工作做个总结；二是想将自己多年来在ARM v8架构操作系统领域的经验分享给大家。希望对大家有所帮助。

感谢自己投身于自己热爱的行业，也感谢自己坚持了下来。

# 基础篇

本篇主要讲解ARM架构操作系统的一些基本的概念，包括软件、硬件方面的概念。描述每个模块的功能职责，以及为什么需要这些模块。

## 启动原理

操作系统一般是由启动程序启动的，或者说是引导程序。启动程序对于操作系统而言非常重要，如果没有启动程序，操作系统根本无法运行。操作系统上运行那么多功能强大的软件更是无从谈起。那么启动到底对操作系统做了什么呢？

目前比较流行的操作系统，如linux可以由uboot引导启动或者IPL启动，但这仅仅是找到操作系统的入口，或者传递了一些配置资源给操作系统，以便操作系统可以根据该资源做一些配置。在找到操作系统入口之前的这些启动程序，不在本书的讲解范围内。本书主要讲解操作系统本身启动过程的启动原理，讲解启动过程，到底做了什么事。

其实对于操作而言，其启动过程主要是为操作系统准备物理环境，开启当前硬件平台的一些硬件特性。初始化一些硬件外设，是否开启缓存cache、内存管理单元MMU等。

试想操作系统在一个未初始化的硬件平台上，应该准备什么样的环境，操作系统才能正常运行起来。下面将从硬件和软件两方面进行说明。

硬件准备

CPU核、内存、IO交互设备，这是一个操作系统能运行起来的最基本的硬件需求。那么所以启动是需要一个CPU核来运行指令的，这个第一个启动的核一般称为主核。其次是内存，系统一般使用的是DDR易失性随机存储器，这个一般由启动程序初始化好。最后是交互设备，一般使用串口，输出一些启动信息，使得用户看得见系统的启动过程。

软件准备

软件需要准备些啥？前面cpu已经跑起来了，会执行代码定义和变量吧，会调用函数吧，有可能发生异常吧。所以，软件层面，需要准备函数调用和分配临时变量的栈，动态申请释放内存的堆。系统的异常向量表需要提前设置好，否则如果启动过程中发生异常或者中断，系统将会因为找不到代码执行路径而出现系统崩溃或其他未知的问题。导致系统无法正常运行下去。

## 中断

中断其实也算是一种异常，一般叫做异步异常。为了便于区分和学习中断，本书将单独分配一个章节讲解中断。那什么是中断？中断存在意义又是什么？在本章节我们将得到答案。

试想这样一个场景：你有一个快递将要送到你的家里，但是你并不知道具体的时间。那你是就待在家里一直等待，还是可以先出去做点其他事情然后让快递员给你打电话呢？我们可以认为，你正在外面（附近）做事，比如买菜什么的，接到快递员的电话，然后你放下手中的事，回去取快递。这样子的一个场景，可以称之为中断场景。当然，至于你取了快递又去不去继续买菜，还是去做其他的事情，就要根据具体情况具体分析。这又和中断返回时是否有其他优先级更高的事件场景相似，这个将在后文讲解。

上述场景仅仅是让大家初步认识，什么是中断。中断带来的好处有什么？

系统可以正常执行其他任务，而不必空转死等中断事件的到来。提高了系统的运行效率。

在ARMV8架构中，有一个用于管理中断源的控制器，叫做General Interrupt Controller通用中断控制器，简称GIC。ARM将中断源分为三类：

SGI，软件触发中断。比如，核间中断。

PPI，私有外设中断。比如，CPU的物理定时器和虚拟定时器。

SPI，共享外设中断。比如，GPIO、I2C、EMMC等外设发生的中断。

ARMV8的GIC V2版本支持1024个中断源，将这些中断源分类对应到上述类别：

0~15是SGI，16~31是PPI，32~1024是SPI。其中0~31是BANKED，即在每个CPU上都有一个独立的副本，每个CPU只能读走自己的副本，不能影响其他CPU上的中断信息。而SPI不一样，他是共享的，只要有一个CPU识别出中断源，该中断源就会清位，另一个CPU就无法再识别该中断源了。

## 异常

## 物理内存

## 虚拟内存

## 地址空间

## 进程管理

操作系统中，进程比线程拥有更大的资源范围。通常一个进程包含，数个线程、当前线程运行的地址空间、已映射的物理地址、当前进程的环境变量、已开启的文件信息等等。所以进程切换将会比线程切换的开销大得多。

## 线程管理

操纵系统中，线程是最小的调度单位。一个线程的正常运行至少需要使用通用寄存器、栈指针寄存器、状态寄存器。因此不同线程之间的切换，需要切换通用寄存器、栈指针寄存器和状态寄存器。由于在进行线程切换时，使用中的通用寄存器会被按序压入栈中，切换回来的时候，这些通用寄存器会被弹栈恢复，因此未使用的通用寄存器就可以不必压栈保存。

## 调度器

从线程管理章节可知，线程是最小的调度单位。做线程切换时，最小切换内容是：通用寄存器和栈（一般指栈基址）。做切换工作的角色主要是调度器完成的。除了做轻量级的线程任务切换，调度器还需要做进程任务切换，这是操作系统中比较常见的切换常见，也称为上下文切换。通常上下文切换，指从当前任务执行环境切换到另一个任务的执行环境，这个任务可以是同进程的线程，也可以是不同进程的任务，甚至有可能是不同虚拟机内的任务。

这里主要讲解一下进程级的上下文切换，一般会切换些什么内容。

通用寄存器x0~x30

浮点寄存器V0~V31

一些状态寄存器，如SPSR

用户态转换表基址寄存器TTBR0\_EL1和TTBR0

线程ID寄存器，TPIDRXX

地址空间ID，ASID

## 同步机制

## 驱动

# 进阶篇

主要将结合实际代码来操作理解操作系统中的概念。

Armv8相关寄存器

主要包含了31个通用寄存器和几个特殊寄存器。如SP，LR等。

cache介绍

MMU介绍

MMU全称，Memory manage unit内存管理单元，管理物理内存映射到虚拟地址空间的地址、权限等。通过MMU可以实现多进程任务之间的虚拟地址空间隔离，使得每个进程任务都满足当前虚拟地址空间中，只有它一个进程任务。

用户态与内核态

调试异常