# 设备树

## 字符设备驱动方式

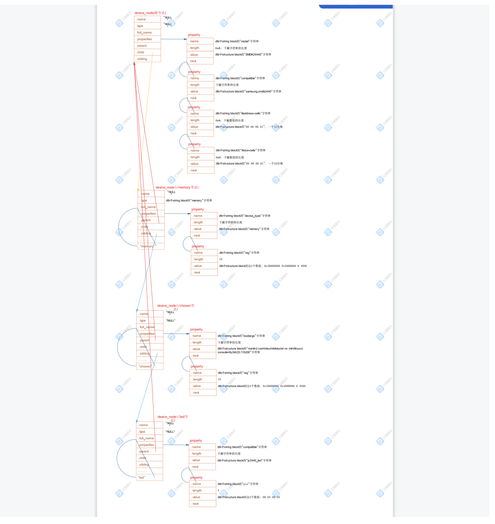
- 传统方法
- 总线设备驱动 bus
  - dev —— 指定硬件资源
  - drv —— 分配设置注册file_operation
- 设备树
  - device tree 其实就是把硬件资源不放到c文件
  - dts 硬件
  - drv

## 规范

- dts
  - 值
    - < > 32位数据
    - " " 字符串
    - [ ] 字节序列 16进制表示一个或多个字节
  - 同一级别名字不能一样 可以用name@3000 等区别
  - 默认值
    - compatible
    - address-cells
    - size-cells
    - ....
  - dts 会包含dtsi文件
- dtb 由dts文件生成

## 内核对设备树支持

- bootloader
  - R0 一般为0
  - R1 machine_id 设备树没用到这个
  - R2 一般设置ATAGS或 DTB 首地址
- head.s head-common.S
  - bootloader -> kernel 处理
    - a. __lookup_processor_type : 使用汇编指令读取CPU ID, 根据该ID找到对应的proc_info_list结构体(里面含有这类CPU的初始化函数、信息)
    - b. __vet_atags : 判断是否存在可用的ATAGS或DTB
    - c. __create_page_tables : 创建页表, 即创建虚拟地址和物理地址的映射关系
    - d. __enable_mmu : 使能MMU, 以后就要使用虚拟地址了
    - e. __mmap_switched : 上述函数里将会调用__mmap_switched
    - f. 把bootloader传入的r2参数, 保存到变量__atags_pointer中
    - g. 调用C函数start_kernel
  - R1 -> __machine_arch_type
  - R2-> __atags_pointer
- 平台信息处理
  - a. 设备树根节点的compatible属性列出了一系列的字符串,表示它兼容的单板名,从"最兼容"到次之
  - b. 内核中有多个machine_desc,其中有dt_compat成员, 它指向一个字符串数组, 里面表示该machine_desc支持哪些单板
  - c. 使用compatile属性的值, 跟每一个machine_desc.dt_compat比较,成绩为"吻合的compatile属性值的位置",
    成绩越低越匹配, 对应的machine_desc即被选中
- dtb转化为device_node



- device_node -> platform_device
  - 设备树的信息最后会传给驱动程序probe函数使用
  - a. 内核函数of_platform_default_populate_init, 遍历device_node树, 生成platform_device
  - b. 并非所有的device_node都会转换为platform_device
    只有以下的device_node会转换:
  - b.1 该节点必须含有compatible属性
  - b.2 根节点的子节点(节点必须含有compatible属性)
  - b.3 含有特殊compatible属性的节点的子节点(子节点必须含有compatible属性):这些特殊的compatilbe属性为: "simple-bus","simple-mfd","isa","arm,amba-bus"
- 操作函数
  - device_node
    - of.h // 提供设备树的一般处理函数, 比如 of_property_read_u32(读取某个属性的u32值), of_get_child_count(获取某个device_node的子节点数)
    - of_address.h // 地址相关的函数, 比如 of_get_address(获得reg属性中的addr, size值)
    - of_match_device(从matches数组中取出与当前设备最匹配的一项)
    - of_dma.h // 设备树中DMA相关属性的函数
    - of_gpio.h // GPIO相关的函数
    - of_graph.h // GPU相关驱动中用到的函数, 从设备树中获得GPU信息
    - of_iommu.h // 很少用到
    - of_irq.h // 中断相关的函数
    - of_mdio.h // MDIO (Ethernet PHY) API
    - of_net.h // OF helpers for network devices.
    - of_pci.h // PCI相关函数
    - of_pdt.h // 很少用到
    - of_reserved_mem.h // reserved_mem的相关函数
  - platform_device
    - of_platform.h // 把device_node转换为platform_device时用到的函数,
      // 比如of_device_alloc(根据device_node分配设置platform_device),
      // of_find_device_by_node (根据device_node查找到platform_device),
      // of_platform_bus_probe (处理device_node及它的子节点)
    - of_device.h // 设备相关的函数, 比如 of_match_device
- 根文件系统查看设备树
  - a. /sys/firmware/fdt // 原始dtb文件

    hexdump -C /sys/firmware/fdt

    b. /sys/firmware/devicetree // 以目录结构程现的dtb文件, 根节点对应base目录, 每一个节点对应一个目录, 每一个属性对应一个文件
  - c. /sys/devices/platform // 系统中所有的platform_device, 有来自设备树的, 也有来有.c文件中注册的
    对于来自设备树的platform_device,
    可以进入 /sys/devices/platform/<设备名>/of_node 查看它的设备树属性
  - d. /proc/device-tree 是链接文件, 指向 /sys/firmware/devicetree/base

## u-boot对设备树的支持

- u-boot启动内核命令
  - bootm <ulmage_addr> // 无设备树,bootm 0x30007FC0
  - bootm <ulmage_addr> <initrd_addr> <dtb_addr> // 有设备树
- 修改dtb文件
  - 方式一 原dtb文件修改
    - 修改属性值方法
      - 新值: newlen (假设newlen > len)
      - a. 把原属性val所占空间从len字节扩展为newlen字节:
        把老值之后的所有内容向后移动(newlen - len)字节
      - b. 把新值写入val所占的newlen字节空间
      - c. 修改dtb头部信息中structure block的长度:
        size_dt_struct
      - d. 修改dtb头部信息中string block的偏移值: off_dt_strings
      - e. 修改dtb头部信息中的总长度: totalsize
    - 增加新属性值
      - a. 如果在string block中没有这个属性的名字,
        就在string block尾部添加一个新字符串: 属性的名
        并且修改dtb头部信息中string block的长度: size_dt_strings
        修改dtb头部信息中的总长度: totalsize
      - b. 找到属性所在节点, 在节点尾部扩展一块空间, 内容及长度为:
        TAG // 4字节, 对应0x00000003
        len // 4字节, 表示属性的val的长度
        nameoff // 4字节, 表示属性名的offset
        val // len字节, 用来存放val
      - c. 修改dtb头部信息中structure block的长度: size_dt_struct
      - d. 修改dtb头部信息中string block的偏移值: off_dt_strings
      - e. 修改dtb头部信息中的总长度: totalsize
  - 方式二 通过u-boot

## 中断相关概念

- 老内核中断与新内核中断不一样，特别是linux4以后，中断号之前是固定的，当中断多以后，这种方式便不适用，新的是哪里空闲就去哪里