Lab3

一、实验目的

本实验旨在通过解析 `virt.dtb` 文件，理解设备树（Device Tree）在嵌入式系统中的作用和结构。设备树作为一种用于描述硬件的数据结构，能够在不修改内核源码的前提下，实现对硬件信息的灵活配置和传递，特别适用于 ARM 架构等设备种类繁杂的系统。

通过本实验，具体目标包括：

- 掌握设备树的基本概念和用途；
- 学习 `.dtb`（Device Tree Blob）文件的结构；
- 实践使用编程语言（如 Python、C 等）对 `.dtb` 文件进行解析；
- 深入理解设备信息如何通过设备树在系统启动阶段传递给内核；
- 为将来在使用 bootloader（如 U-Boot）和内核集成设备树做准备。

二、实验过程

本实验通过手动解析设备树文件（`virt.dtb`），理解其内部结构，掌握设备树的基本格式和关键字段。实验使用 Python 编写程序，具体过程如下：

1. 准备设备树文件

首先需要准备好一个设备树的二进制文件 `virt.dtb`，该文件通常由 QEMU 虚拟平台生成，也可以从官方提供的设备树编译器（dtc）工具编译获得。

2. 编写 Python 脚本读取 DTB 文件头

设备树的起始部分是一个头部结构 `fdt_header`，包含了设备树中各个部分的偏移与大小信息。

```python
FDT_HEADER_FORMAT = ">10I"
FDT_HEADER_SIZE = struct.calcsize(FDT_HEADER_FORMAT)
```

- `">10I"` 表示 “大端（`>`）10 个无符号 32 位整数（`I`）”，正对应规范中头部的十个字段：magic, totalsize, off_dt_struct…size_dt_struct
- `FDT_HEADER_SIZE` 计算了头部总字节数（10×4 = 40 字节）。

PYTHON

```
# 结构块中 token 常量
FDT_BEGIN_NODE = 0x00000001
FDT_END_NODE   = 0x00000002
FDT_PROP       = 0x00000003
FDT_END        = 0x00000009
```

- 定义结构块中各 token 的数值。规范第 5.4.1 节列出了这五种 token 及其含义：
  - `FDT_BEGIN_NODE`：节点开始，后跟节点名字符串
  - `FDT_END_NODE`：节点结束
  - `FDT_PROP`：属性定义
  - `FDT_END`：结构块结束。

PYTHON

```
def read_header(f):
    # 读取头部的原始二进制数据
    # `blob` 是 **Binary Large OBject**（二进制大对象）的缩写。这个词广泛用于编程中，表示一段原始的二进制数据
    blob = f.read(FDT_HEADER_SIZE)
    # `struct.unpack` 将二进制解压成十个 Python 整数。
    hdr = struct.unpack(FDT_HEADER_FORMAT, blob)
    # 给每个字段命名，方便后续引用
    keys = [
        "magic", "totalsize", "off_dt_struct", "off_dt_strings",
        "off_mem_rsvmap", "version", "last_comp_version",
        "boot_cpuid_phys", "size_dt_strings", "size_dt_struct"
    ]

    # 按照规范中字段顺序，用 `keys` 列表打包成字典，方便后续通过名字访问。例如 `header["magic"]` 得到 `0xd00dfeed`。
    return dict(zip(keys, hdr))
```

3. 解析结构块（Structure Block）

结构块是设备树的核心部分，里面通过不同的 token（标记）描述树状结构，包括节点（Node）和属性（Property）信息。每个节点用 `FDT_BEGIN_NODE` 开头，用 `FDT_END_NODE` 结束。

PYTHON

```python
def parse_structure_block(f, offset, size, max_nodes=2):
    f.seek(offset)  # 定位到结构块起始位置
    end = offset + size
    nodes = []

    while f.tell() < end:
        token, = struct.unpack(">I", f.read(4))  # 读取当前 token

        if token == FDT_BEGIN_NODE:
            # 读取节点名称（以 null 结尾的字符串）
            name_bytes = bytearray()
            while True:
                b = f.read(1)
                if b == b"\x00":
                    break
                name_bytes += b
            name = name_bytes.decode()

            # 对齐到 4 字节边界
            pad = (4 - ((len(name_bytes)+1) % 4)) % 4
            f.read(pad)

            nodes.append(name)  # 收集节点名称
            if len(nodes) >= max_nodes:
                break

        elif token == FDT_PROP:
            # 如果是属性，跳过属性长度和名字偏移
            length, nameoff = struct.unpack(">II", f.read(8))
            f.read(length)  # 跳过属性值
            pad = (4 - (length % 4)) % 4
            f.read(pad)  # 对齐

        elif token in (FDT_END_NODE, FDT_NOP:=0x4):
            # 空操作或结束节点，无需处理内容
            continue

        elif token == FDT_END:
            break  # 整个结构块结束
```

```
        else:
            raise ValueError(f"未知 token {token:#x}")

    return nodes
```

4. 编写主函数并执行解析流程

PYTHON

```
def main():
    with open("virt.dtb", "rb") as f:
        header = read_header(f)

        # 校验 magic 值是否合法（设备树固定魔数）
        assert header["magic"] == 0xd00dfeed, "magic 不匹配"

        # 打印头部的关键信息
        print("DTB Header:")
        for k in ("totalsize", "off_dt_struct",
"size_dt_struct"):
            print(f"  {k} = {header[k]}")

        # 解析结构块前几个节点名称
        print("\nStructure Block 前几个节点名称: ")
        nodes = parse_structure_block(
            f,
            header["off_dt_struct"],
            header["size_dt_struct"],
            max_nodes=3  # 最多打印 3 个节点名称
        )
        for n in nodes:
            print("  -", n)
```

执行该脚本将输出设备树的头部信息和结构块中的几个顶层节点名，例如：

```yaml
DTB Header:
  totalsize = 1048576
  off_dt_struct = 64
  size_dt_struct = 6920

Structure Block 前几个节点名称:
  -
  - psci
  - memory@40000000
  - platform@c000000
  - fw-cfg@9020000
  - virtio_mmio@a000000
  - virtio_mmio@a000200
  - virtio_mmio@a000400
  - virtio_mmio@a000600
  - virtio_mmio@a000800
```

三、心得体会

通过本次实验，我对设备树在嵌入式系统中扮演的角色有了更清晰的认识。设备树的设计理念在于将硬件描述从内核代码中分离出来，这种方式极大地提高了系统的可移植性和灵活性，也减少了内核开发过程中的重复劳动。

虽然实验中我们使用 QEMU 直接加载内核而未通过 bootloader，因此未实际使用设备树传参的功能，但通过对 `virt.dtb` 文件的解析，我进一步理解了设备树中的层次结构、属性键值以及它如何表达一个完整的硬件系统。

在解析过程中，我也遇到了一些挑战，比如文件格式的二进制结构较为复杂、字段的对齐与偏移等问题。不过这些问题也让我意识到了解底层实现的重要性。在调试和查看设备树内容的过程中，我还接触到了一些工具（例如 `dtc` 命令行工具），这些都为后续深入学习嵌入式开发打下了基础。

总的来说，这次实验不仅让我掌握了设备树的基本操作，也提升了我对底层硬件抽象机制的理解，非常有收获！