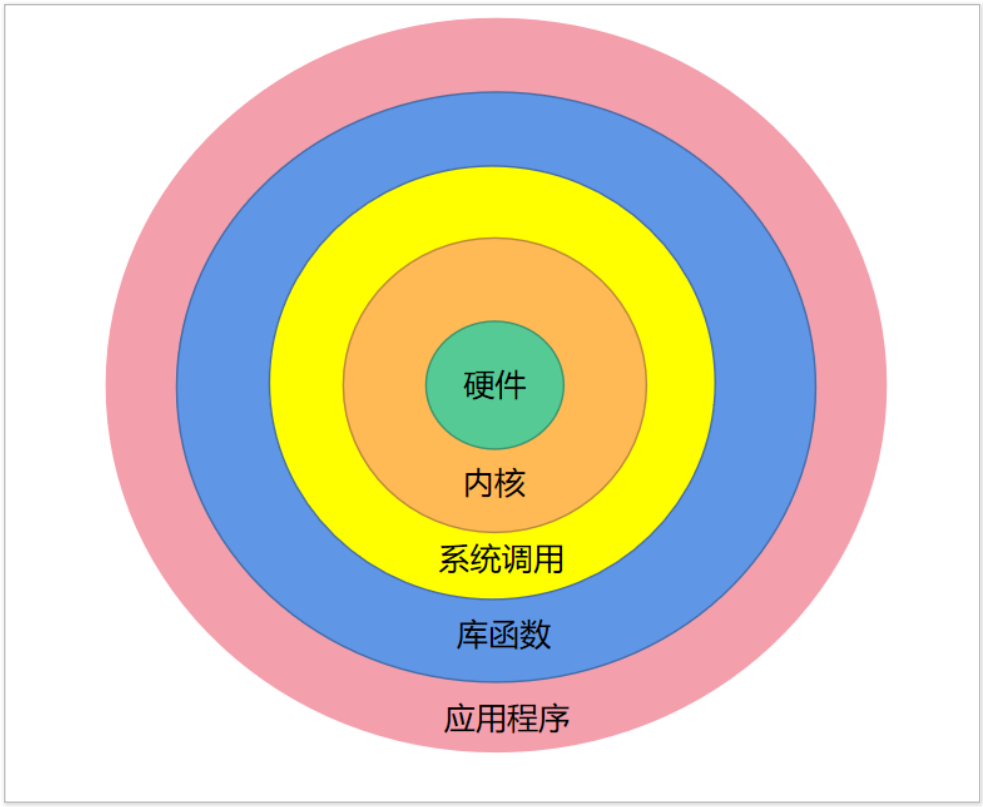
1.1 操作系统与 Linux io 框架_物联网 / 嵌入式工程师 - 慕课网

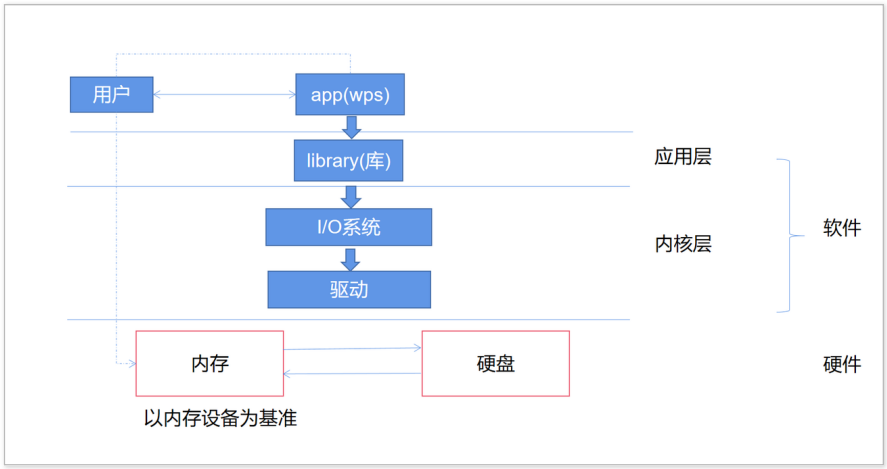
> 慕课网慕课教程 1.1 操作系统与 Linux io 框架涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

- io 描述的是硬件设备之间的数据交互，分为输入 (input) 与输出 (output)
    - 输入: 应用程序从其他设备获取数据 (read) 暂存到内存设备中
    - 输出: 应用程序将内存暂存的数据写入到其他设备 (write)
- 操作系统通常包含两种不同的含义
    - 第一种含义: 指完整的软件包：包括
      核心软件
      与
      应用软件
        - 应用软件: 命令解释器, 图形用户界面, 文件操作工具与文件编辑器)
        - 核心软件: 管理和分配计算机资源 (即 cpu,RAM, 其他设备)，即操作系统核心软件 (内核)
    - 第二种含义: 专指操作系统核心软件 (内核)

- 内核的职责如下:
    - 进程管理：分配 cpu 资源，用于执行程序指令
    - 内存管理：如今计算机内存容量可谓相当可观, 但软件的规模也保持了相应的增长, 内存资源仍然属于有限资源, 内核必须公平与高效的管理内存资源, 其中虚拟内存管理技术
    - 文件管理：允许对文件执行创建，获取，更新以及删除
    - 设备管理：计算机外界设备可实现计算机与外部世界的通讯
    - 联网管理：使计算机可以进行网络通讯
    - 提供应用编程接口 (API): 进程可利用内核入口点请求内核区执行各种任务
- 一般分为 用户层与内核层
    - 用户层：表示在内核层之上的 库 (如 glibc) 与 应用程序 (app)
    - 内核层：操作系统内核
- 用户层与内核层是相辅相成，用户层的应用程序依赖于库或者内核, 库与内核给应用层提供服务
- 内核通过系统调用来给应用层提供接口



- 系统调用 是 Linux 内核提供给应用程序的访问接口, 当需要 Linux 内核提供服务时, 则需要访问系统调用
- 库函数 为了实现某个功能而封装起来的 API 集合, 能够提供统一的编程接口，更加便于应用程序的移植
- glibc 是属于 GNU(GNU's Not unix) 工程的一部分, 这个工程当初的目标是为了开发一款完整的操作系统, 但在开发过程中将除了 Linux 内核以外的组件都开发完成, 由于难度很大, 开发周期长, 在 1992 年 由 Linus Torvalds 开发出来了 Linux 内核, 填补了 GNU 系统的一个重要空白, 所有后面将 GNU 组件与 Linux 合并组成现在的 GNU/Linux
- glibc 包含 标准 c 库函数集合 和 系统调用
    - 标准的 c 库函数是跨平台的，既可以在 Linux 系统下调用, 也可以在 windows 系统下调用
    - 系统调用是 Linux 内核给用户提供的访问接口, 但在 glibc 中封装了系统调用接口而形成了 glibc 的库函数.
    - glibc 库函数主要是封装了系统调用的过程, 相应的系统调用一般实现在 Linux 内核中
    - 一般的 glibc 中的库函数都会与系统调用关联，但也有库函数不需要使用系统调用，比如字符串操作函数.
- Linux io 框架也是分层设计, 这里以将内存中的数据存储到硬盘中为例



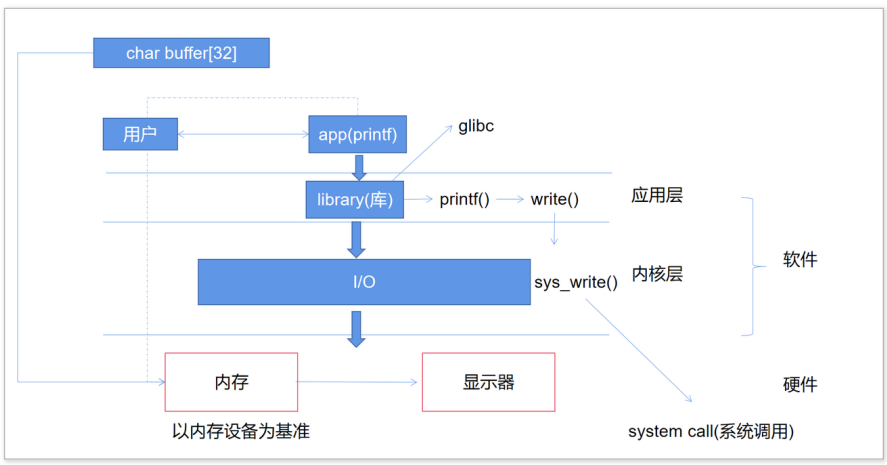
- 应用程序通过调用 操作系统提供的 io 接口（函数）向内核进行 io 请求，由内核最终完整相应的 io 操作

- Linux io 框架基于
  一切皆文件
  的 思想来设计
  - 目的：屏蔽底层不同设备之间的 io 差异，给应用层提供统一的操作接口
  - 思想：即将底层的 io 操作统一抽象成文件操作，操作提供系统只需要提供一组文件 io 操作接口就可以为应用程序提供 io 服务

- 文件 io 操作主要包含：
  - open：打开
  - close：关闭
  - read：读取
  - write：写入
  - lseek：定位

- 文件 io 接口的设计本身来沿用了人的 操作习惯
  - 大脑相当于 内存设备
  - 书籍或者其他笔记本相当于另一个设备
  - 读书：相当于大脑获取数据（read）
  - 写字：相当于将大脑数据写入到其他存储介质中

- 下面以 printf io 过程为例来说明



- printf io 的过程本质上是将暂存在内存中的数据写入到显示器中
  - printf 函数首先会调用 glibc 中 write 函数来发出 io 请求
  - write 函数在通过调用由操作系统内核提供的系统调用 sys_write 函数最终完成 io 操作

- 下面是 sys_write 系统调用在内核中的实现

```
SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
        size_t, count)
{
    struct fd f = fdget_pos(fd);
    ssize_t ret = -EBADF;

    if (f.file) {
        loff_t pos = file_pos_read(f.file);
        ret = vfs_write(f.file, buf, count, &pos);
        if (ret >= 0)
            file_pos_write(f.file, pos);
        fdput_pos(f);
    }

    return ret;
}
```

- 练习：
    - 1. 理解 操作系统 io 框架
    - 2. 理解 一切皆文件的设计思想

全文完