学生入门任务

与大家简单介绍下 linux io stack 的情况

- 一些资料主要包括
  - netstorage.md
    - 这个是我根据舒老师网络存储课程记得一些笔记，主要需要理解一些存储技术的发展，是如何从**集中式**（主要就是 `DAS/NAS/SAN` ）转到**分布式**，能够帮助大家从更**宏观的角度切入**并且能够清楚的理解我们**将要聚焦**的部分是什么
  - io.html
    - 简单的梳理IO流程，并没有整理到可以发表的规整程度，而且包含了很多的**代码分析**
      - 可以先过过，有些关键词的解释啥的
    - 里面有关键的**几幅图**，IO栈的概览图，**先装在脑子里**
  - iostack.drawio.html
    - 自己画的一个iostack的理解图，主要包括了以下内容
      1. io stack中io相关数据结构
      2. PCIe配置空间/CPU SPA空间/内存地址空间/VM空间的一顿map关系
- **其它相关**名词与基础知识
  1. PCIe子系统
     - `pcie.md`
  2. DMA
     - 当前啊，DMA是PCIe设备上的一个硬件单元，有访问**主机全部主存**的能力，你告诉它怎么搬运数据，它就搬运了
  3. CPU SPA空间
  4. SAS/SATA基础知识
     - `sas.md`

我们所聚焦的存储，简单概述一下

- 详见netstorage.md
  1. 无论是分布式的还是集中式的，最后都要聚焦到**存储设备（盘）**上，或者说最后都会归到linux kernel的storage IO栈上
     - 比如分布式文件系统（GFS/...）在各个节点上大都需要依赖单机的文件系统（ext4/...），而更高级的存储组件甚至会基于分布式文件系统继续构建，或者基于其它等等
  2. 而在集中式存储中，例如NAS，存储设备是直接连在 **瘦服务器（详见netstorage.md）** 上，依然依赖的是linux kernel的IO栈，就是说要你们装在脑子里的那个图
- 所以我们就先聚焦我们的主要工作位置，理解它在**无论哪个存储时代**中所扮演的**不可或缺**的角色，关于IO栈大部分资料都是基于它展开的，包括 `io.html/iostack.drawio.html` 等等
- **这里再一句话把IO stack概括一下**
  - IO就是把**内存**中**从某个地方开始**的**指定长度**的数据**搬入（I）或搬出（O）到存储设备（最常见的就是各种盘）上某个地方开始**的**指定长度**的**位置**
    - 内存暴露给我们的是**物理地址**，线性的
    - 盘也一样，在我们的眼里它是一个**一维数组**，专业的名字叫做lba，logic block address，说白了就是一个个的块，这也为什么我们总说块设备
- **从下到上**看一下各个层次暴露给上层的接口是什么

1. 首先是**盘**，以**NVMe盘（PCIe接口）为例，盘这个东西很简单，简单来说就响应读写操作**
    - 告诉**盘**我要读**哪里开始**的**多少数据**，**盘上的DMA就给我送到主机内存了**
    - 告诉**盘**我要写**哪里开始**的**多少数据**，**盘上的DMA就主动从主机内存把数据写下去了**
2. 上面说的那个动作的**实现位置**大都是各个**硬件**对应的驱动
    - 所以盘的上一层就是驱动
3. 驱动继续向上，驱动暴露给上层的**接口**也很简单
    - 驱动接收的是上层单个的IO请求
        - 一个**单独的IO请求**就是**把数据从哪里搬到哪里，搬多长**
    - 可以看到驱动这里并不关系IO请求是啥意思（论文还是代码，数据还是元数据）
        - **驱动层是没有语义的**
4. 驱动再向上一般是到块层，上层请求被抽象成IO请求后，并不一定能最高效的发挥出盘的性能，而且**不同的应用**也可能落到**同一个盘上**
    - 所以在这层是不是可以尝试调度一下啊，太大的IO分割一下，相邻的IO合并一下，是不是就凸显出这层的价值来了
5. 再向上，就要到文件系统了
    - 如果把块层的接口直接暴露给用户，是不是显得有些奇怪，用户编程的时候还要考虑把什么东西放到哪里，这个是不是很低效
    - 所以聪明的程序员是不是就要再抽象一层出来了，把这些公共的内容自己来管理，暴露更加简单的接口给上层，所以文件系统就来了
6. 通常来讲啊，文件系统就是**管理盘**的，上接用户的请求，下面看看怎么把恰当的块分配给这个请求，分配元数据数据啥的，而且元数据数据需要一致，所以我们说文件系统就有了语义
    - 元数据数据要保持一致性是不是就要同时落盘成功呢，有啥策略呢，所以文件系统的设计上就会有事务类的操作，典型设计
        - 因为**元数据**与**数据**可能就是**两个IO**，这两个IO一般是在一个事务中的
7. 文件系统继续向上，一个系统中可能有多个盘，多种文件系统，甚至有些伪文件系统它不是真正需要落盘的文件系统，就是为了构造一个树形的结构方便用户使用的，比如/proc文件夹等等
    - 为了把这些都统一起来，也是linux一切皆文件的思想，所以上面还有一层VFS，这个就是我们在linux使用过程中看到的这个tree结构，不同的文件目录可能就会落到不同的后续的实际文件系统中
8. 到此为止，上面就是用户常用的read/write接口了，差不多就是怎么个意思

布置下任务 —— 有的人做过的就不用做了

*	task	描述与需要掌握到的程度
1	过一下这些文档，其中有代码的部分**先不细究**， 先别在这里画太多时间	过个眼熟
2	写一个内核模块（内核模块的hello word）	熟悉Makefile与内核编写的入门
3	把第二步的makefile提交到git@github.com:doubleDDDD/practise.git	熟悉git的基本操作
4	在之前内核模块的基础上，向**系统注册一个字符设备**， 在 `dev` 与 `sys` 目录下找到它， 并尝试**读写**该**字符设备**看看会发生什么 （需要自己实现**该字符设备的读写接口**）	理解dev与sys目录存在的意义

*	task	描述与需要掌握到的程度
5	除了读写接口之外，利用**ioctl**操作一下第4步中的**字符设备**	理解ioctl的用法，感受用户态与内核态的交互
6	保留上述字符设备，再创建一个设备注册到platform总线； 并且向platform总线注册一个驱动； 使自己的驱动能够驱动刚才的设备	理解驱动与设备以及总线的关系， 在**sys或dev目录**下找以上**总线设备驱动的存在**
7	把这个跑起来 `git@github.com:LeapIO/debug_kernel.git`	这是qemu模拟的一个内核调试环境， **熟悉linux内核的编译与使用**
8	在kernel的main函数 `start_kernel` 处断点，实现单步调试	熟悉gdb的基本使用，断点/info/bt等等
9	第4步中所写的驱动直接放到qemu的内核源码树中**编译**并且**跑起来**	熟悉linux kernel的Makefile与Kconfig的作用， 主要就是编译选项
10	利用断点追一下上述驱动中所调用内核的接口， 主要是 `add/register` 类的接口	回顾一下gdb，更好的理解**设备总线驱动模型**
11	在qemu中的os下跑一个最基本的用户程序 `hello word` ， 用户态的hello word，**任务很简单但是涉及到的内容比较多**	**理解静态可执行文件与动态可执行文件， 以及动态库的link，以及ramdisk是什么**
12	在qemu中，执行 `make dk` （详见**说明文档**）， 它会在qemu下模拟出一个**sata磁盘**， 对该磁盘做**ext4文件系统**或**不做**， 如果做了文件系统就进去创建文件，读写文件操作一下， 如果不做文件系统的话就用**其它方式**访问一下这个设备	回顾下用户态能触发的基本的IO操作
13	到此为止，你拥有了一个完整的可debug的IO执行环境， 你的所有操作实际上你都可以**利用gdb追到**， 这里我要求大家总结出一个**基本调用栈关系来**， 每一层的关键函数追到就可以， **从用户态到最后驱动触发IO的那个地方**，主要层包括**vfs/ 具体文件系统/通用块层/SCSI上中下三层/disk**	这个能过一下， 对IO栈就会有一个比较全面的认知了
14	IO栈清楚之后**聚焦**到我们现在关注的**驱动层**，也是距离**硬件** 最近的一层，之前在platform bus的例子中应该已经清楚了**总线设备驱动**模型。 现在我们来**思考PCIe总线**。我们的例子是ahci驱动， 位于 `drivers/ata/ahci.c` ，这里有3个对象， **ahci驱动/ahci控制器/PCIe bus**， 把ahci设备的注册与驱动的注册的位置找到并立即	理解PCIe子系统，PCIe设备什么时候枚举， 寻址方式是什么，什么是配置空间， MMIO是什么，DMA是怎么work的
15	gdb追ahci驱动与设备配对后的初始化过程	理解sata驱动的实现， 理解sda等盘符是怎么被创建的