Lab5 驱动

本次实验中，我们会实现一个简易的 sd 卡驱动，并了解 MBR 分区格式。

使用如下命令获取本次实验的代码：

```
git fetch --all
git checkout lab5
git merge lab4 # or else
# Handle Marge Conflicts
```

1 I/O 框架

本节抽象地介绍了简易的基于队列的异步 I/O 框架。

1.1 异步读写

对于块设备驱动，往往需要实现对其的读写操作，而由于 I/O 速度较慢，为了避免阻塞当前 CPU，我们可以采用异步的方式来实现，具体思路如下

- 读：向设备发送读请求和目标地址，然后等待中断直到设备完成读取，再读取设备返回的数据
- 写：向设备发送写请求、目标地址以及待写入的数据，等待中断直到设备完成写入

当然在这其中我们还需要额外考虑许多其他的细节，例如我们在发送读写请求时会设置对应的标志位，在中断结束后我们需要清除这些相应的标志位，来避免重复的中断，同时，如果设备发生错误中断，我们也需要重新执行当前操作等。

1.2 请求队列

对于多个进程均想使用 I/O 的情况，则需要一个 buf 请求队列。进程作为生产者，每次请求都将一个 buf 入队，然后阻塞（睡眠）直到驱动来唤醒它。而 I/O 驱动作为消费者，每次取出并解析队首的 buf，向设备发出具体的读写操作，完成后唤醒对应的进程。

1.3 习题一

请补全 `src/common/buf.h` 以便于 SD 卡驱动中请求队列的实现，即每个请求都是一个 buf，所有请求排成一队。

队列的实现可以参考 `src/common/list.h` 中 `Queue` 与 `queue_` 系列函数。

2 块设备驱动

在块设备中，我们会用到如下几个简单的术语：

- 块大小：每个块的大小，通常为 512 B
- 逻辑区块地址（Logical block addressing, LBA）是从 0 开始的块下标，即第几块

对于内核的其他模块而言，异步的块设备驱动通常包括三个函数，初始化函数如 `sd_init`、设备读写请求函数如 `sdrw`、设备中断处理函数如 `sd_intr`。设备的初始化比较硬件，暂且略过，让我们从读写函数开始讲起。

2.1 读写请求

`sdrw` 是给内核其他模块（如文件系统）使用设备的接口，其接受一个 buf，然后进行类似 1.1 中的步骤，我们会用 `wait_sem\post_sem` 来实现等待的过程。

2.2 设备中断

设备中断的到来意味着 buf 队首的请求执行完毕，于是我们会根据队首的请求类型进行不同的操作

- 读请求，合法的中断说明数据已从设备中读出，我们将其从 MMIO 中读取到 buffer 的 data 字段上即可
- 写请求，合法的中断说明数据已经写入了设备

然后清中断 flag，并继续进行下一个 buf 的读写请求（如果有的话）

2.3 习题二

请完成 `src/core/sd.c` 中的 `sd_init`，`sdrw`，`sd_intr`，然后分别在合适的地方调用 `sd_init` 和 `sd_test` 完成 SD 卡初始化并通过测试。

3 制作启动盘

现代操作系统通常是作为一个硬盘镜像来发布的，而我们的也不例外。但在制作镜像时，需要注意遵守树莓派的规则，即第一个分区为启动分区，文件系统必须为 FAT32，剩下的分区可由我们自由分配。为了简便，我们采用主引导记录（Master boot record, MBR） 来进行分区，第一个分区和第二个分区均约为 64 MB，第二分区是根目录所在的文件系统。简言之，SD 卡上的布局如下

```
 512B            FAT32       Your fancy fs
+-----+-----+-------------+----------------+
| MBR | ... | boot partion | root partition |
+-----+-----+-------------+----------------+
 \   1MB   / \    64MB    / \     63MB     /
  +-------+   +----------+   +------------+
```

3.1 MBR

MBR 位于设备的前 512B，有多种格式，不过大同小异，一种常见的格式如下表

Address	Description	Size (bytes)
0x0	Bootstrap code area and disk information	446
0x1BE	Partition entry 1	16
0x1CE	Partition entry 2	16
0x1DE	Partition entry 3	16
0x1EE	Partition entry 4	16
0x1FE	0x55	1
0x1FF	0xAA	1

但这里我们只需要获得第二个分区的信息，即上表中的 Partition entry 2，这 16B 中有该分区的具体信息，包括它的起始 LBA 和分区大小（共含多少块）如下

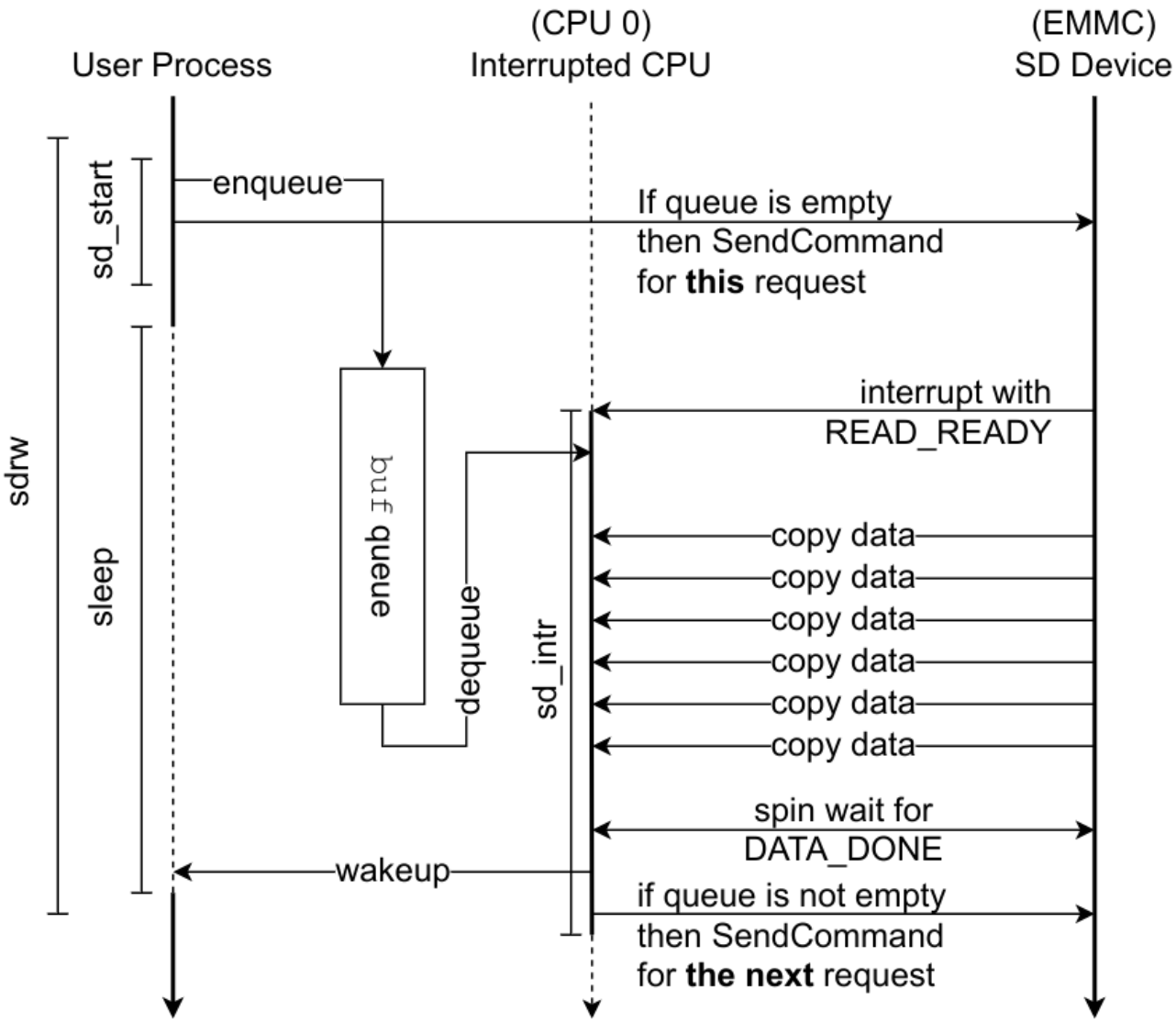
Offset (bytes)	Field length (bytes)	Description
...	...	...
0x8	4	LBA of first absolute sector in the partition
0xC	4	Number of sectors in partition

3.2 习题三

请在 `sd_init` 中解析 MBR 获得第二分区起始块的 LBA 和分区大小以便后续使用

4 提示

4.1 sd读示意图



4.2 sd_start

效果是程序向sd卡设备预约工作。

不是要求完成的部分，但需要阅读其中的细节，去了解sd.c的工作流程。

- `write = b->flags & B_DIRTY`
- `cmd = write ? IX_WRITE_SINGLE : IX_READ_SINGLE;`
- `resp = sdSendCommandA(cmd, bno)`

- ```
  int done = 0;
  u32* intbuf = (u32*)b->data;
  if (write) {
      // Wait for ready interrupt for the next block.
      if ((resp = sdWaitForInterrupt(INT_WRITE_RDY))) {
          PANIC("* EMMC ERROR: Timeout waiting for ready to write\n");
          // return sdDebugResponse(resp);
      }
      asserts(!*EMMC_INTERRUPT, "%d ", *EMMC_INTERRUPT);
      while (done < 128)
          *EMMC_DATA = intbuf[done++];
  }
  ```

4.3 sd.c 代码注释

略

5 提交

提交：将实验报告提交到 elearning 上，格式为 `学号-lab5.pdf` 。

从lab2开始，用于评分的代码以实验报告提交时为准。如果需要使用新的代码版本，请重新提交实验报告。

截止时间：2022年10月26日 19:30。逾期提交将扣除部分分数。

报告中可以包括下面内容

- 代码运行效果展示（测试通过截图）
- 解析 MBR 获得第二分区起始块的 LBA 和分区大小
- 实现思路和创新点
- 对后续实验的建议
- 其他任何你想写的内容

报告中不应有大段代码的复制。如有使用本地环境进行实验的同学，请联系助教提交代码（最好可以给个git仓库）。使用服务器进行实验的同学，助教会在服务器上检查，不需要另外提交代码。
  ```