Swap 挑战性任务

任务说明

> 注：**本任务实现应能通过课下 Lab6 的测试。** 评测逻辑不对本题实现思路做出限制，只会尝试分配超过物理内存大小的内存。为了通过测试，你的实现需能保证系统至少可以分配出比物理内存多 4MB（1024 页）的内存。

该任务需要同学按照给定要求为 MOS 实现内存物理页交换（swap）功能。即当 MOS 当前缺少空闲页时，能够将某些页面写入到磁盘中，从而产生新的空闲页以供分配，并且不影响对被换出页的访问。具体而言，该任务可分为如下部分： - 页面的换出（swap out）：当内存紧张时将某页内容换出到磁盘中，将该页标记为空闲页。 - 页面的换入（swap in）：当进程需要访问被换出的页面时，能够将该页面从磁盘中换入到内存中。

参考文档

- PIIX4 文档
- PIO 文档

任务要求

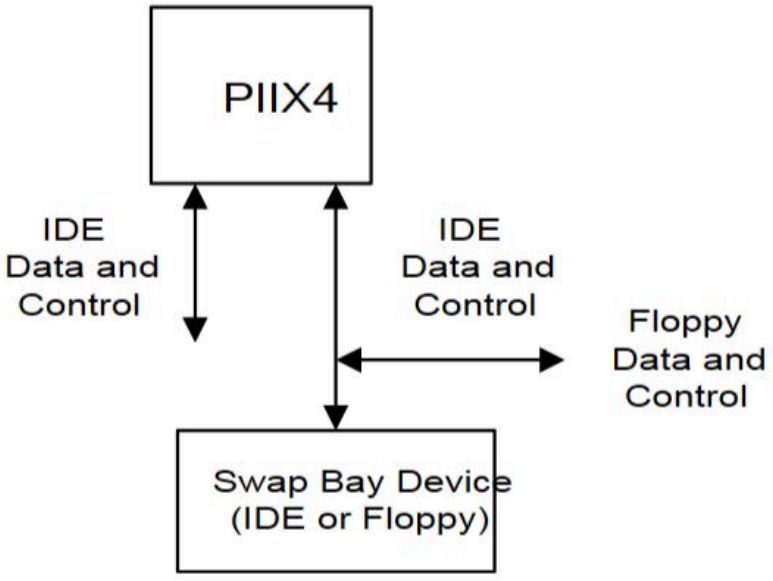
- 完整可运行的内核代码。
- 内核文档（请**务必详细地介绍你的实现**）。
- 查重。

具体内容

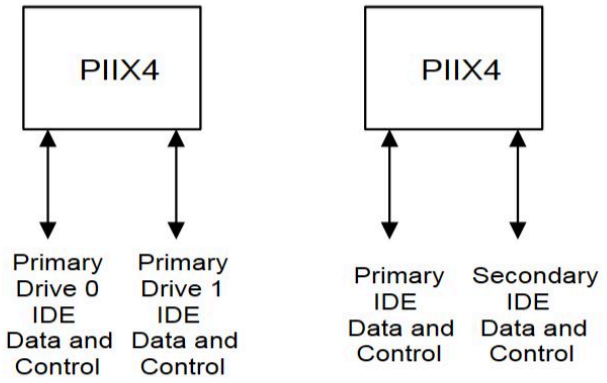
下面给出一套可能的实现方案，同学们可以根据自己的理解进行实现或修改。

PIIX4 简介

MOS 基于 Malta 开发板，而 Malta 所使用板载的 PIIX4 芯片组支持 IDE 等设备并且拥有两个独立的 IDE 通道。



每个 IDE 通道连接着一个 IDE 控制器，每个控制器可以连接两块 IDE 设备，故 MOS 中的 PIIX4 最多支持 4 块 IDE 设备。



两个IDE控制器的偏移地址分别为：

- 主控制器命令块偏移: `0x1F0`
- 从控制器命令块偏移: `0x170`

以上偏移基于 PIIX4 的 PCIIO 基地址 `0x18000000`，也即 MOS 中定义的 `MALTA_PCIIO_BASE`。

MOS 使用 PIO 模式来读写 IDE 磁盘，使用的设备寄存器如下:

Table 28. IDE Legacy I/O port definition: COMMAND BLOCK (CS1x# chip select)

IO Offset	Register Function (Read/Write)	Access
00h	Data	R/W
01h	Error/Features	R/W
02h	Sector Count	R/W
03h	Sector Number	R/W
04h	Cylinder Low	R/W
05h	Cylinder High	R/W
06h	Drive/Head	R/W
07h	Status/Command	R/W

读写 IDE 磁盘的操作可以参考 MOS 原有实现，如对 PIO 感兴趣可以参考PIO 文档，此处不再赘述。

为了避免与文件系统使用的主 IDE 控制器冲突，我们可以在内核态使用从 IDE 控制器读写磁盘，用于存储交换页面。

可以修改 `Makefile` 文件中定义的 `QEMU_FLAGS` 变量来挂载多块磁盘：

```
QEMU_FLAGS              += -cpu 4Kc -m 64 -nographic -M malta \
                        $(shell [ -f '$(user_disk)' ] && echo '-drive
id=ide0,file=$(user_disk),if=ide,format=raw') \
                        $(shell [ -f '$(empty_disk)' ] && echo '-drive
id=ide1,file=$(empty_disk),if=ide,format=raw') \
                        $(shell [ -f '$(swap_disk)' ] && echo '-drive
id=swap,file=$(swap_disk),if=ide,format=raw') \
                        -no-reboot
```

按照挂载顺序 0, 1 号 IDE 磁盘被主控制器使用，而 2 号 IDE 磁盘 `swap_disk` 则被从控制器使用，`swap_disk` 文件的创建可以参考 `empty_disk`。

交换机制

交换缓存

为了避免频繁的 IO 操作，提高系统性能，需要预留一组页面实现交换缓存。当页面从主存换出时，先将其换出到交换缓存中。当页面需要再次被换入主存时，如果页面在交换缓存中，则直接使用缓存中的数据，无需从磁盘中读取该页面。

如果需要换出页面但交换缓存已满，需要使用随机算法或 LRU 算法等 Cache 置换算法，选定缓存中的页面并将其写入磁盘中。

- `xorshift`

  算法：一种通过位操作进行的伪随机数算法，主要使用异或和移位来混合种子值，生成新的伪随机数，并更新种子值。以下是

  `xorshift`

  的示例代码：

  ```
  #include <stdint.h>
  uint32_t xorshift32(uint32_t state) {
      uint32_t a = 14;
      uint32_t b = 17;
      uint32_t c = 6;
      state ^= state << a;
      state ^= state >> b;
      state ^= state << c;
      return state;
  }
  ```

如果需要生成某一范围的随机数，可以在此基础上使用线性同余限定范围。LRU 算法的实现可直接参考教材。

页面换出与换入

当内核中内存消耗完全时，需要使用之前提到的随机算法或者 LRU 算法选取某一可交换页面换出。在这一过程中，需要更新所有使用被选中物理页的页表，修改其权限位将其标志为 Swap 页面，清空所有对应该页面的 TLB 与 Cache。在对应进程访问该换出页面时，需要触发 TLB 异常从而能够换入该页面。在换入后同样需要修改该页面对应的所有页表项，恢复相应权限位以表明该页面已被换入。

注意事项

- 内核所使用的页面（`freemem` 以下的页面）与进程页表等页面应作为保留页不能被换出。
- 需要支持 4MB（1024 页）的交换内存，也即总共可用 68MB 内存。交换内存少于 4MB 时可能无法通过测试。

测试

完成该任务后，需在课程网站附带文档提交完整可运行的 MOS 代码。我们会创建若干进程消耗内存以测试交换机制的实现情况。