

v1.5.1存在的问题和Makefile使用说明

李荣 2023/11/8——v1.0

本版本基于v1.3.22虚拟机版本构建，在v1.3.22虚拟机版本上引入了v1.3.20和v1.3.22真机版本的修改，并改进makefile，统一了虚拟机和真机，可以在不修改内核代码的情况下，通过调整makefile的参数，来选择运行在真机或者虚拟机上。同时该版本修复了部分warning。考虑到该版本将真机和虚拟机统一了，我们决定将该版本定为v1.5.1。

本文档首先会介绍Makefile使用方法。然后简单介绍一下该版本引入的修改，因为大部分修改已经在相关文档中有详细说明了，这里只是简单介绍一下，重点在于本版本在测试中发现的问题。

Makefile使用说明

下图33行到52行为makefile可配置的参数，文件中每个参数都给出的详细的说明，这里不再赘述。需要注意的是，当选择orangepfs作为启动分区的文件系统格式的情况下，会强制启动分区和根文件系统同一个分区，这个makefile会检查 `BOOT_PART_NUM` 和 `ROOT_FS_PART_NUM` 是否相等。

若选择使用grub启动，1号分区作为启动分区时，`GRUB_CONFIG` 需要选择 `boot_from_part1.cfg`；2号分区作为启动分区时，`GRUB_CONFIG` 需要选择 `boot_from_part2.cfg`；其他情况需要自己编写grub配置文件。同时还需要注意，在真机上使用grub引导启动，目前还存在问题，进入shell后无法响应键盘中断。

```
25 # Programs, flags, etc.
26 ASM      = nasm
27 DASM     = ndisasm
28 CC       = gcc
29 LD       = ld
30 AR       = ar
31
32 #added by sundong 2023.10.28
33 #####用户可以输入的变量#####
34 #选择OS的启动环境, virtual 代表虚拟机 (qemu), real 代表真机
35 MACHINE_TYPE = virtual
36 #安装的硬盘, 例如真机启动时该变量可能为 /dev/sda; 虚拟机启动无需设置此变量
37 INS_DEV=/dev/sda
38 #启动分区的分区号, 数字类型, 例如: 1
39 BOOT_PART_NUM=1
40 #根文件系统所在的分区号, 数字类型, 例如: 2
41 ROOT_FS_PART_NUM=2
42 #用于区分是使用grub chainloader
43 #可选值为 true false
44 USING_GRUB_CHAINLOADER = false
45 #grub 安装的分区, 数字类型, 例如: 5
46 GRUB_PART_NUM=5
47 #选择启动分区的文件系统格式, 目前仅支持fat32 和orangepfs
48 BOOT_PART_FS_TYPE= fat32
49 #grub的配置文件, 提供了一个默认的grub配置文件, 配置为从第1块硬盘分区1 引导
50 GRUB_CONFIG=boot_from_part1.cfg
51 #使用虚拟机时虚拟镜像的名称, 该虚拟镜像应该放在hd/ 文件夹下
52 BOOT_IMG=virtual_disk.img
53 #####
```

makefile提供以下命令：

`make clean`：清除生成的二进制文件。

`make`：编译内核、`os_boot`、`loader`和用户程序。

`make install`：安装操作系统。

引入的修改

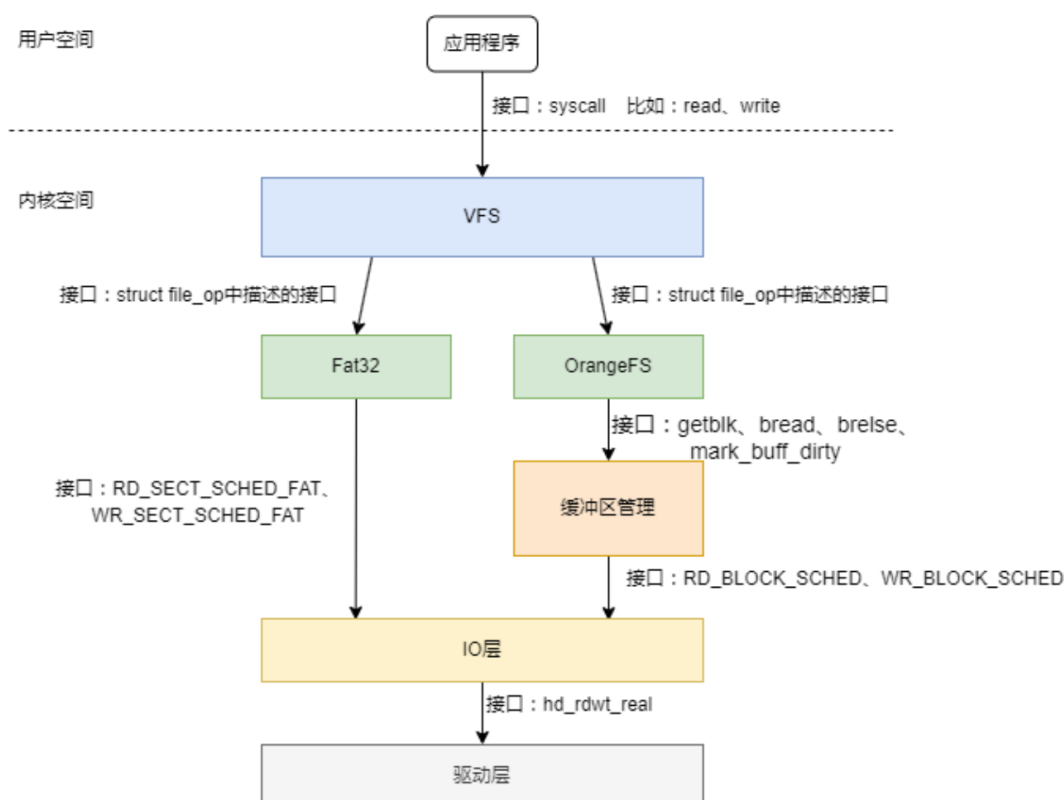
1、slab内存分配地址对齐

在真机上，部分寄存器要求传入的值按照 2^n 对齐，比如在ahci设备初始化中，`port_rebase()` 中为 `port->clb` 分配了1K大小的内存空间，该变量映射到了sata的寄存器，要求起始地址1k对齐，但slab分配的内存并不是对齐的，硬件会自动将地址向低地址对齐，导致对寄存器读写改变了其他地址的数据。修改后，slab分配的内存能够按照 2^n 对齐。详细修改过程和原理请参考文档 [v1.3.20真机bug修改.pdf](#)。

2、完善SATA驱动和中断处理函数

在真机上，原有的SATA中断处理函数不完善，只检测了一种错误类型，将其它情况都认为是读写完成。同时再加上不合理的清中断顺序，导致驱动无法正确读写磁盘。修改后，驱动解决了这个问题，但错误处理并未完善，我们对于可处理的错误，仅进行重启端口来解决错误。该部分详细改动请参见文档 [v1.3.20真机bug修改.pdf](#)。

同时，为了完善错误处理后重发任务，我们添加了sata硬盘错误标志 `sata_error_flag`，当读写出现错误时，中断处理函数会进行错误处理，完成错误处理后，会将 `sata_error_flag` 置为1，然后唤醒进程。SATA驱动会根据 `sata_error_flag` 的值返回 `TRUE` 或 `FALSE`。注意到22版本IO层次结构及接口，我们考虑后续在IO层完成磁盘读写任务失败后的重新发送，当驱动返回 `FALSE` 后，IO层应该尝试再次执行当前读写任务。



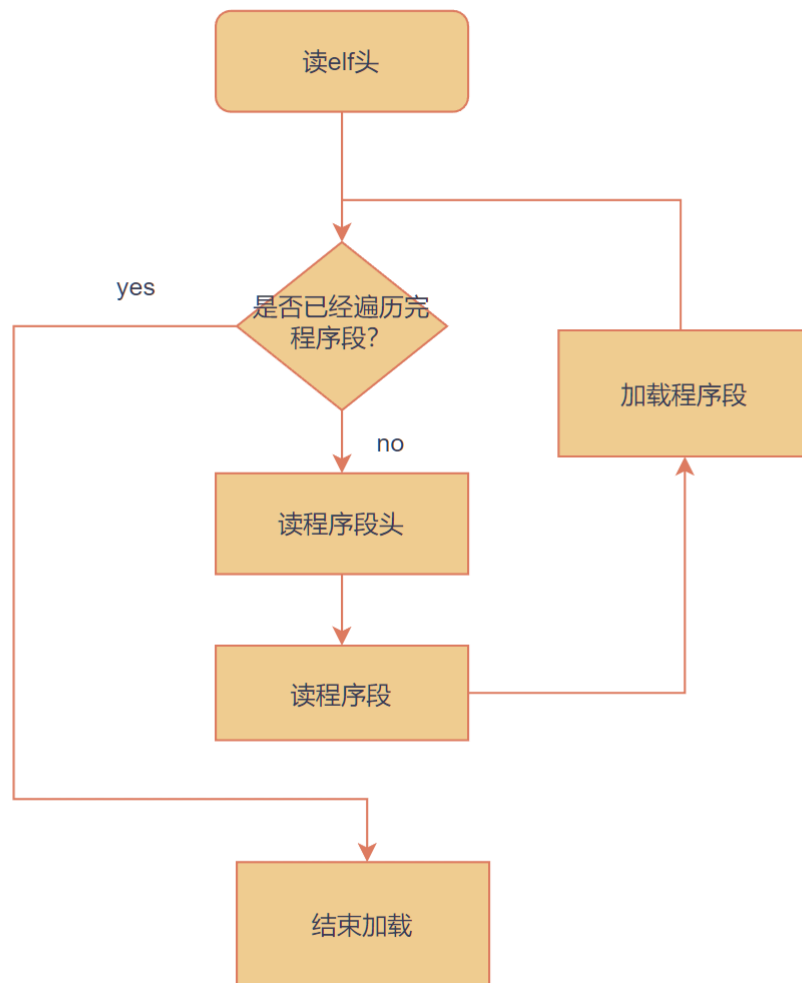
注：

FAT32文件系统未适配缓冲区管理层，因此直接调用的IO层的读写扇区函数。

3、修改loader中的文件读写接口

在真机上，22版本的代码会出现kernel无法正常加载的问题，经过排查，发现loader将kernel文件一次性读入内存后再进行分析，这个过程中kernel文件过大导致缓冲区溢出。我们修改了loader的文件系统接口，删去了原有的 `read_file()`，添加了 `open_file(char *filename)` 和 `read(u32 offset, u32 len, void *buf)` 来实现文件的随机读写。详细细节请阅读文档 [v1.3.22真机bug修改.pdf](#)。

这里记录一个虚拟机 (qemu) 上非常奇怪的bug。在使用 `read()` 来读elf文件的程序段头的时候，我们原先是每一次读一个程序段头，根据程序段头读取程序段，加载程序段；再读下一个程序段头，进行下一个程序段的处理。处理流程如下。



但该流程在虚拟机上存在问题，在第五次读程序段头的时候，qemu会闪退回初始状态，然后启动、加载mbr、os_boot、loader，再闪退回初始状态，陷入这样的循环。该bug在删去qemu的boot menu启动选项（下图28行）后就不会触发。资料显示该启动选项是模拟真机的bios，用来选择系统启动盘的。

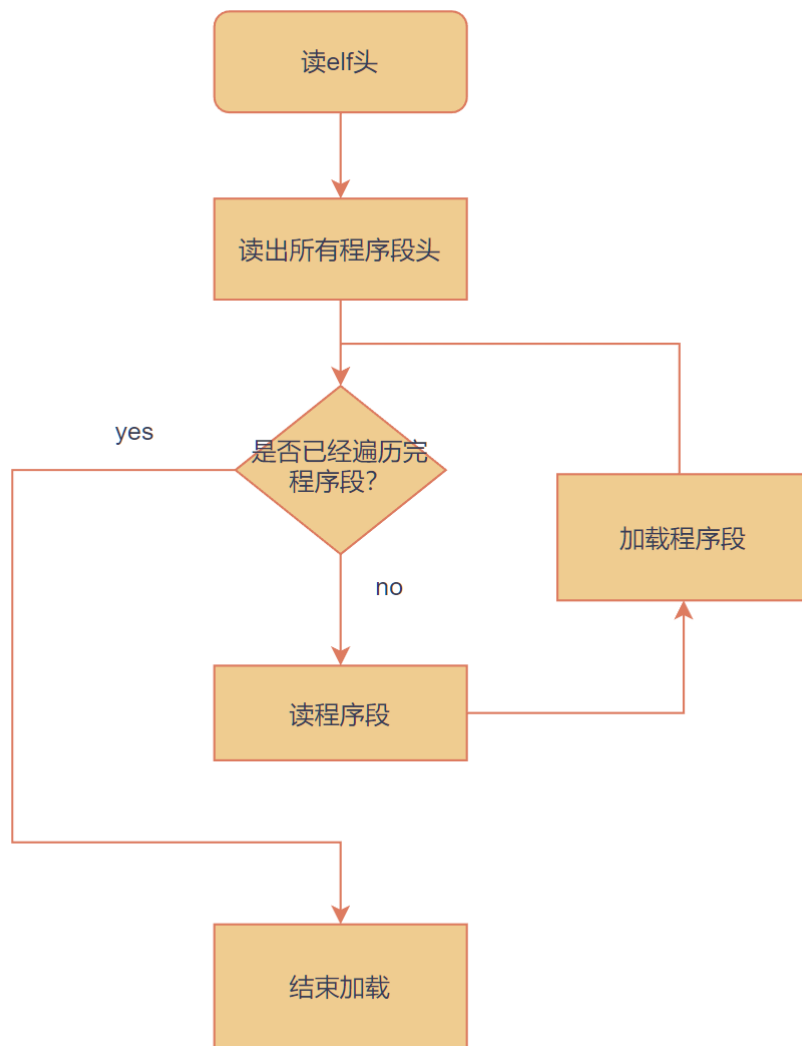
```

25 qemu-system-i386 \
26 -device ich9-ahci,id=xiaofeng \
27 -drive id=disk,file=b.img,if=none -device ide-hd,drive=disk,bus=xiaofeng.0 \
28 -boot menu=on \
29 -monitor stdio

```

根据gdb调试的信息，程序段头非常小（大约32字节），而底层的文件系统实际上是以块（大概4K字节）来读取硬盘的。五次读程序段头实际上调用的参数完全一样，都是读取相同的一个块，区别只有读出后将数据复制到指定地址时，会根据偏移量计算复制的起始地址。但在第五次读程序头时，在底层的读扇区操作发生了闪退。**具体原因不明。该bug只会在开启了boot menu选项的虚拟机（qemu）上出现，真机上没有这个问题。**

考虑到所有的程序段头都是放在一起的，称为程序段头表（*program header table*），且长度较小，我们修改为将程序段头一次性读入。这种情况下虚拟机就不会出问题了。



4、修复文件路径名长度过短的问题

目前orangepfs下文件名长度最长为12字节，路径长度最长为128字节。相关宏定义在os/include/fs.h中。

```
19  #define MAX_PATH 128
20  #define MAX_FILENAME_LEN 12
```

但在文件系统中因为若干函数错误地将 MAX_FILENAME_LEN 当做 MAX_PATH 来使用，导致某些情况下传递的路径名最长仅为12字节。本版本修复了这些错误。

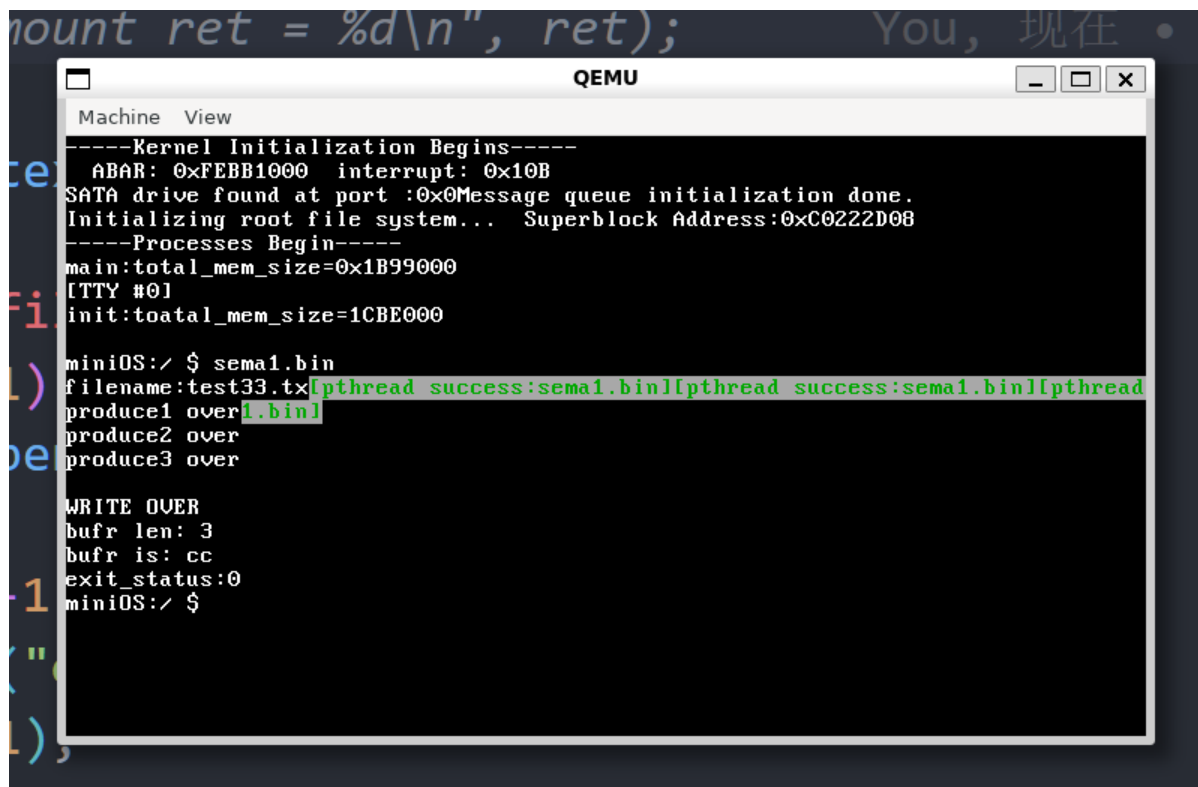
本版本存在的问题

1、多线程读写

测试程序：sema1.bin，创建打开一个orangepfs或fat32文件系统目录下的文件，然后创建三个线程向该文件写入数据。所有线程都结束后，主进程从该文件读出数据。

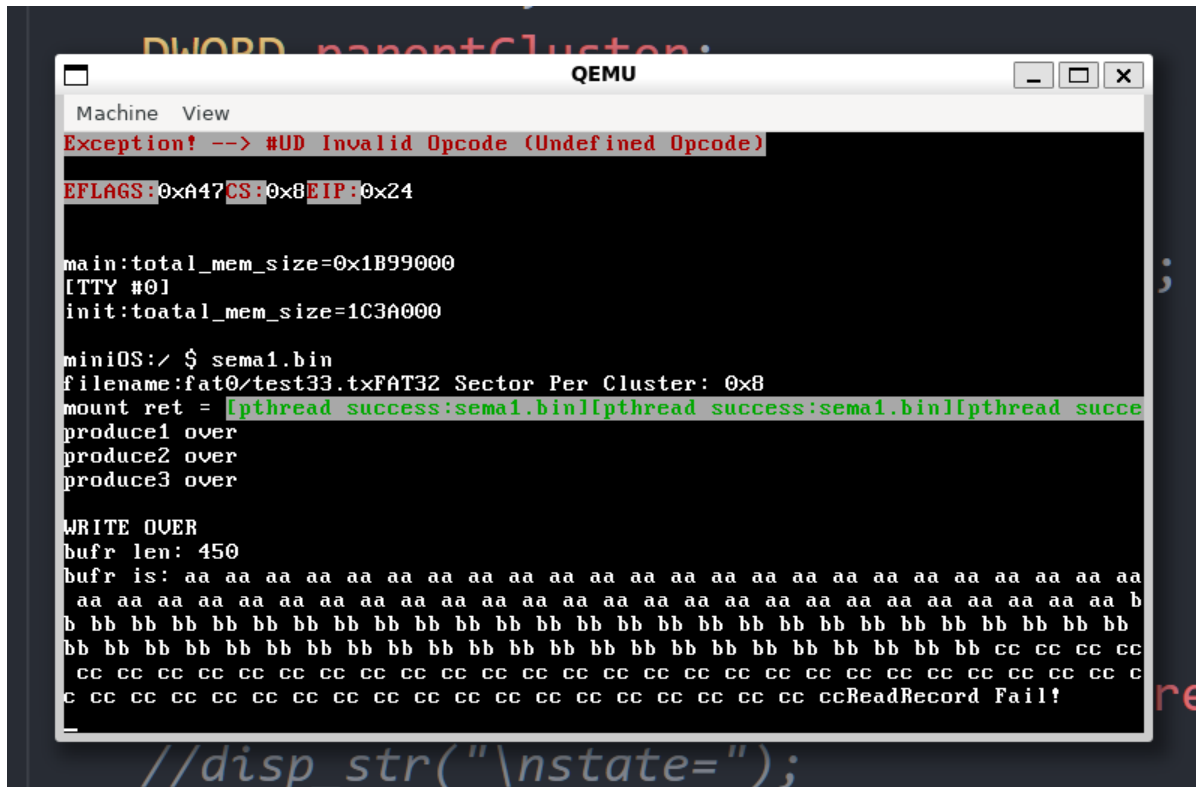
经过测试，在orangepfs下进行多线程读写文件，部分写入的数据会缺失，其余结果都正常。在fat32下进行多线程读写文件，写入的数据完整，但删除文件的函数调用（`unlink()`）会报错。本测试在v1.3.20版本上测试是没有问题的。推测是22版本在orangepfs下添加了缓冲层，部分数据未写回，也可能是线程机制的问题。

下图是在orangepfs下的测试。



```
mount ret = %d\n", ret); You, 现在 •
Machine View
-----Kernel Initialization Begins-----
ABAR: 0xFE8B1000 interrupt: 0x10B
SATA drive found at port :0x0Message queue initialization done.
Initializing root file system... Superblock Address:0xC0222D08
-----Processes Begin-----
main:total_mem_size=0x1B99000
[TTY #0]
init:toatal_mem_size=1CBE000
miniOS:/ $ sema1.bin
filename:test33.tx[pthread success:sema1.bin][pthread success:sema1.bin][pthread
produce1 over1.bin]
produce2 over
produce3 over
WRITE OVER
bufr len: 3
bufr is: cc
exit_status:0
miniOS:/ $
```

下图是在fat32下的测试。在程序的最后，删除文件的系统调用（`unlink()`）导致了异常。



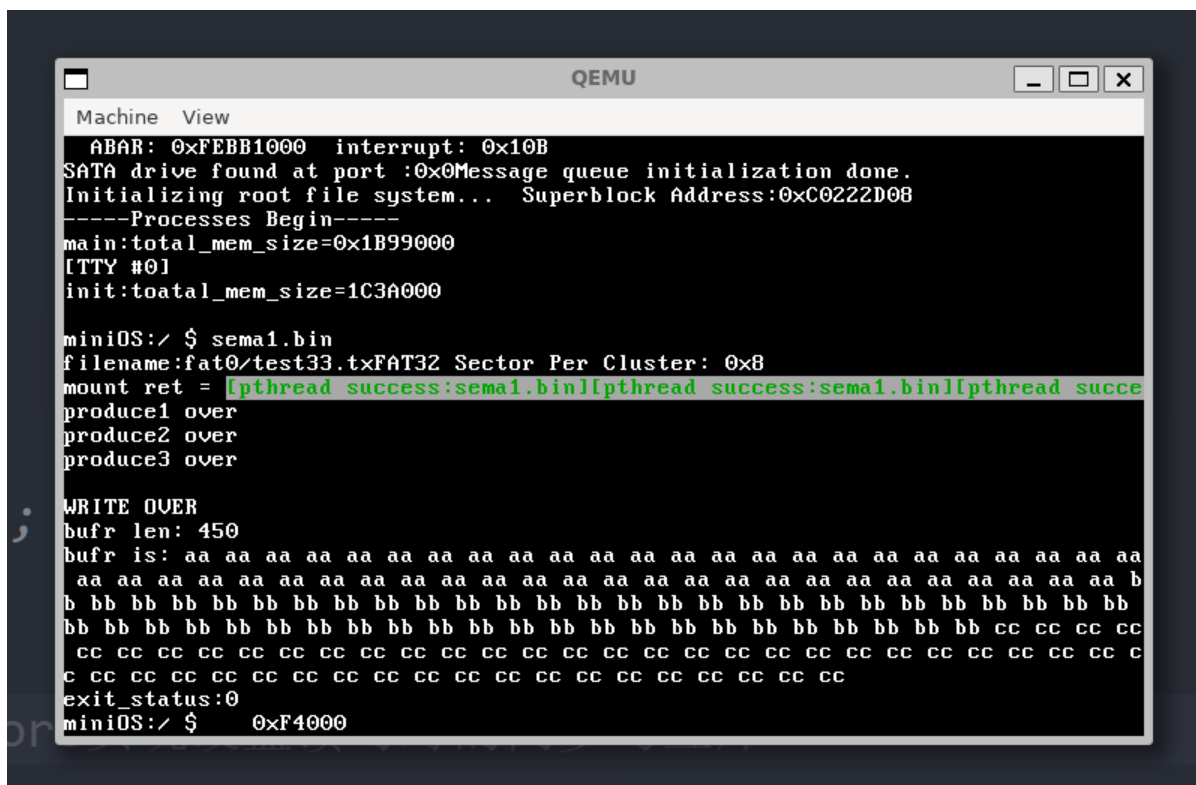
```
Machine View
Exception! --> #UD Invalid Opcode (Undefined Opcode)
EFLAGS:0xA47 CS:0x8 EIP:0x24

main:total_mem_size=0x1B99000
[TTY #0]
init:toatal_mem_size=1C3A000

miniOS:/ $ sema1.bin
filename:fat0/test33.tx FAT32 Sector Per Cluster: 0x8
mount ret = [pthread success:sema1.bin][pthread success:sema1.bin][pthread succe
produce1 over
produce2 over
produce3 over

WRITE OVER
bufr len: 450
bufr is: aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
b bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
c cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
ReadRecord Fail!
```

在fat32下测试，且不调用 `unlink()`。



```
Machine View
ABAR: 0xFEbb1000 interrupt: 0x10B
SATA drive found at port :0x0 Message queue initialization done.
Initializing root file system... Superblock Address:0xC022D08
-----Processes Begin-----
main:total_mem_size=0x1B99000
[TTY #0]
init:toatal_mem_size=1C3A000

miniOS:/ $ sema1.bin
filename:fat0/test33.tx FAT32 Sector Per Cluster: 0x8
mount ret = [pthread success:sema1.bin][pthread success:sema1.bin][pthread succe
produce1 over
produce2 over
produce3 over

WRITE OVER
bufr len: 450
bufr is: aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
b bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
c cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
exit_status:0
miniOS:/ $ 0xF4000
```

[illegible]

在真机上使用grub启动操作系统，能正确加载loader和kernel，能进入shell，但进入shell后，无法响应键盘。经过gdb调试发现，在此过程中，其他中断（0号时钟中断、5号sata中断）都能正常发生，仅有键盘中断（1号中断）无法发生。查看中断控制器i8259A内部的中断屏蔽寄存器（内存地址0x21），键盘中断未被屏蔽。调整真机bios选项、更换键盘usb接口仍然无效。回退到20版本、22版本，发现该问题至少从20版本开始就存在了。

3、缓冲块写回问题

7


```
3175      ar *tsbut = NULL;
(gdb) n
3176      f_head *bh = NULL;
(gdb) p nr_dir_blks
$5 = 1
(gdb) p nr_dir_entries
$6 = 52
(gdb) p bh->count
$7 = 3222986332
(gdb)

kernel.gdb.bin      umount_fat0.sh
launch-qemu-gdb.sh  user
launch-qemu.sh      version.txt
idealist@idealist-QiTianM4500-N000:~/桌面
• _minios$ git branch
```

经过对上面问题的修改后，缓冲块写回仍然存在问题。发现缓冲层写回的逻辑似乎有点问题。调用函数 `sync_inode(struct inode *p)` 并不能保证inode被写回磁盘。该函数的逻辑是找到inode所在的缓冲块，然后将其脏位置为1，最后调用函数 `bre1se()`。而 `bre1se()` 会判断count是否为0，只有 `count == 0` 才会写回缓冲块。这就导致了如果有多个函数使用了该缓冲块（调用了 `bread`），而还没有释放它（调用 `bre1se`），那么即使使用 `sync_inode()` 该缓冲块也不会写回到磁盘。

总结来说缓冲层写回的问题**主要是 `bread()` 和 `bre1se()` 数量不匹配导致缓冲块无法及时写回，同时缓冲层也缺少一个强制写回的函数接口**。目前暂时的修改是限制 `bread()` 函数里的count值增加，即删去下图中的304行。该改动会使得缓冲块较为频繁的写入磁盘。

```
291  buf_head *bread(int dev, int block)
292  {
293      buf_head *bh = getblk(dev, block);
294      // 若used == 1, 说明已经在hash tbl中了, buffer中也有数据了
295      acquire(&bh->lock);
296      if (!bh->used)
297      {
298          // 该buf head是一个新分配的, 此时应该从硬盘读数据进来
299          RD_BLOCK_SCHED(dev, block, bh->buffer);
300          // 标记为已被使用
301          bh->used = 1;
302          bh->count = 1;
303      }else{
304          // bh->count++;
305      }
306      release(&bh->lock);
307      return bh;
308  }
```



```
316 void brelse(buf_head *bh)
317 {
318
319     acquire(&bh->lock);
320     if(bh->count > 0){
321         bh->count--;
322     }
323
324     if(bh->dirty&&bh->count == 0){
325         sync_buff(bh);
326     }
327     release(&bh->lock);
328
329 }
```

DanielSun, 4个月前 • 基本实现了