

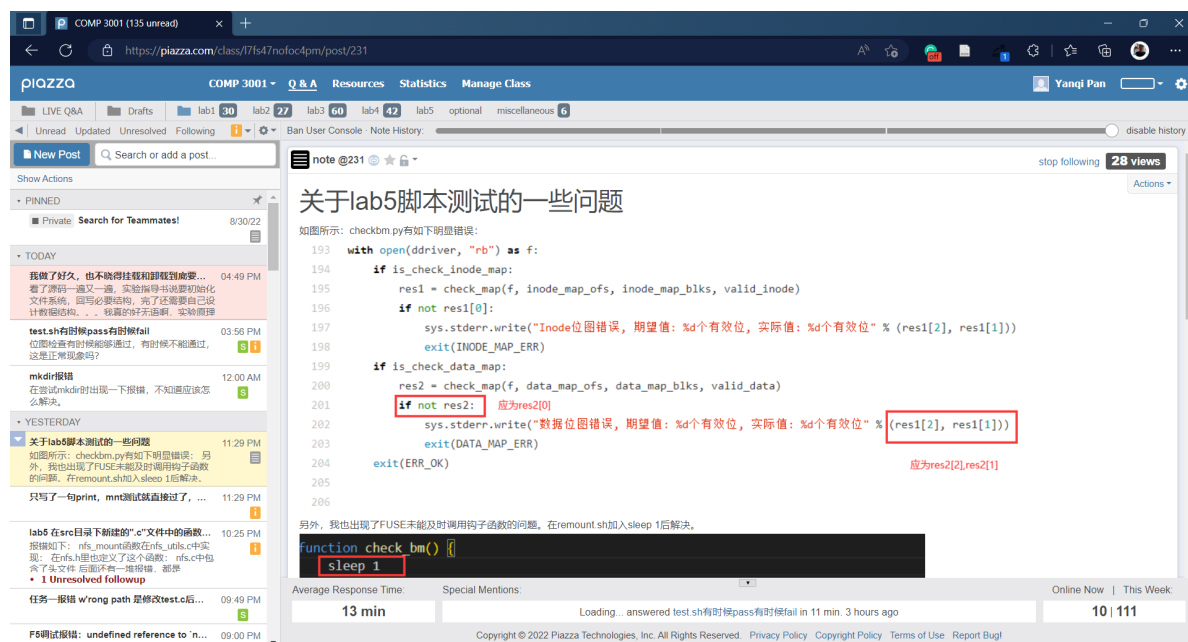
关于OS-lab实验五的补充说明（续）

随着实验的不断进行，有不少同学提出了宝贵的问题，我们就共性问题做出回答，以进一步提高同学们的实验效率，避免进一步踩坑

1. 脚本修正部分

1.1 位图检测脚本 checkbm.py 更新

非常感谢这位匿名同学为我们找到的问题：



修改方法如上图所示。目前这个 BUG 可能会导致一些本来数据位图没有检测通过的同学通过测试。但这是 F-Tutorials 的问题，因此，我们做出两个说明：

- 对于已经通过更正前脚本测试的同学：**你们已经通过，不用再修改脚本**。这里本来我们要检测的是数据位图只有一个有效位，即 `mkdir ./tests/mnt/dir0` 后，根目录 `inode` 会分配一个数据块用于装 `dir0` 的 `dirent`。但是有的已经通过的同学似乎有更多的有效位，虽然不用继续修改，**但要保证在验收过程中给出数据位图有效位个数的理由**。比如如果你的数据位图有两个有效位，那么为什么有两个，是分配了新的数据块吗？为什么要分配这个数据块呢？言之有理即可。
- 对于还未通过更正前脚本测试的同学：**修改脚本以获得正确的错误提示信息，这是为了避免让你们陷入困惑**，因为这个BUG会导致 `checkbm` 错误地告诉你期望数据位图个数为2，下面是修改后的输出结果，即，告诉大家数据位图应该为1（理由见上）。

```
root: case 5.1 - umount /home/students/.../user-land-filesystem/fs/th1fs/tests/mnt
/home/students/.../user-land-filesystem/fs/th1fs/tests/checkbm.py:2: DeprecationWarning: The distutils package is deprecated and slated for removal in Python 3.12. Use setuptools or check PEP 632
for potential alternatives
  from distutils.log import error
数据位图错误，期望值: 1个有效位，实际值: 2个有效位 fail: case 5.2 - check bitmap: 数据位图错误，请使用 checkbm.py 和 ddriver 工具自行检查。注：在命令行输入 ddriver -d 并且安装 HexEditor 插件即可查看当前 ddriver 介质情况

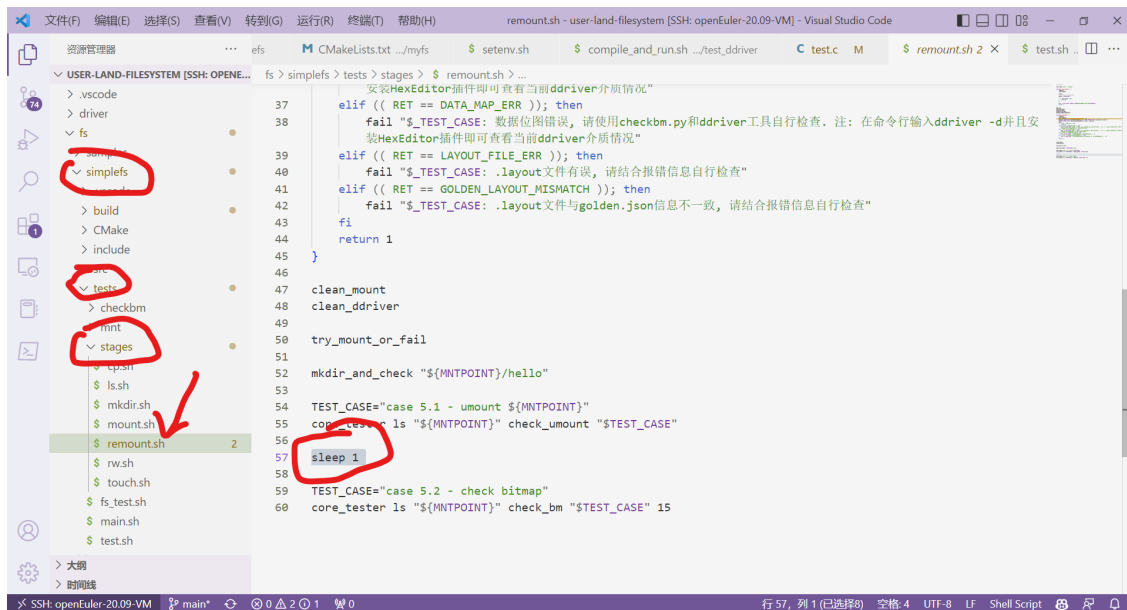
Score: 15/30
fail: 再接再厉! (15/30)
```

1.2 关于 test.sh 时而过时而不过，且报错 remount 部分

这个问题有好多同学提出来，这是个非常好的问题，感谢你们的提出。事实上这个问题与 FUSE 框架、系统状态有关。简单来说就是调用了挂载命令，但 FUSE 还没有真正地去调用我们写的 `destory` 钩子，因此数据没有刷回盘。那么，解决这个问题最好的方法就是调用了 `umount` 后等一段时间再查看 `ddriver` 上的数据。这也是为什么很多同学单独的命令可以通过，但是脚本时好时坏的原因了（即，用户输入延

迟挺大的，数据已经刷完了）。这个问题解决方案有很多种：

- **手动测试**：指导书提交文档部分的命令跑过则通过
- **修改脚本**：以SFS为例，我们建议在 `./tests/stages/remount.sh` 下，`TEST_CASE 5.1` 和 `TEST_CASE 5.2` 间插入 `sleep` 命令，以达到延迟效果。



2. 额外的Tips

2.1 如何更快速地完成本实验？

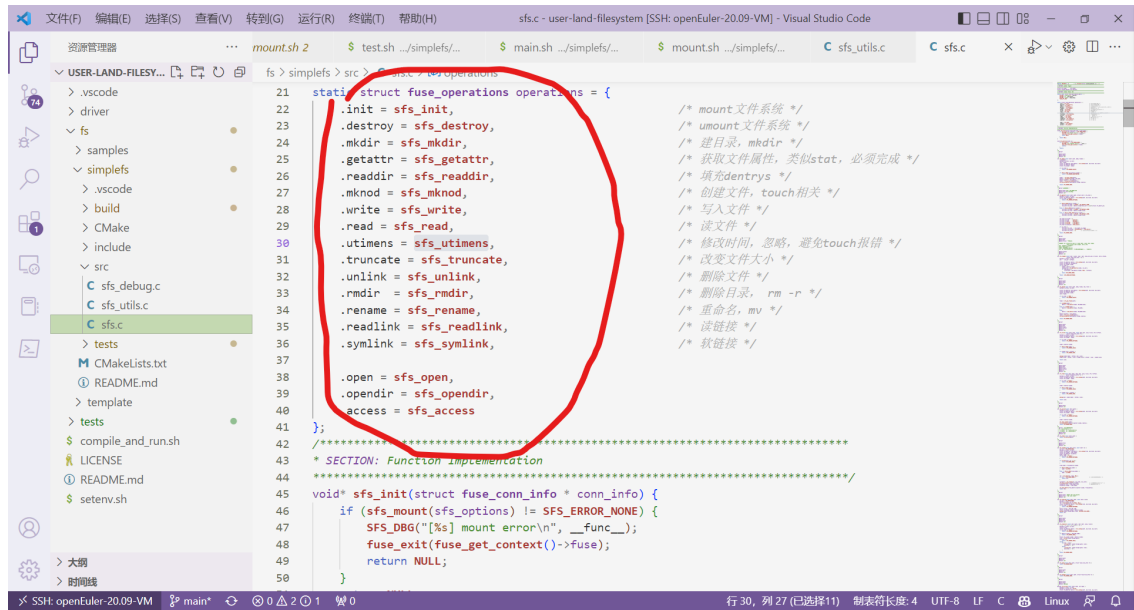
- **术语与概念**：由于理论课并未涉及文件系统的全貌，因此基本概念与术语一定要理解，例如：**挂载（格式化）、卸载**。这在体会真实文件系统的实现上具有重要意义。
- **代码复用**：“好的程序员不写代码，而是看懂后复用代码”，这是一个很经典的论述，这对未来**工作、科研、学习**中也常常适用。因此在 Lab 5 中，**我们非常鼓励**同学复用 SFS 或是其他开源项目的代码进行开发。但这需要大家看懂理解 SimpleFS 的关键部分，知道怎么复用。例如：**SFS如何实现挂载**？我们要求同学们要加入数据位图，但SFS没有，那么怎么在挂载部分修改 SFS 的数据结构和空间布局来**分配、读取数据位图**？
- **介质布局**：`ddriver -d` 命令将介质的布局拷贝到 vs code 当前目录下，一旦发现问题就可以通过 `HexEditor` 来检查。**介质布局**是文件系统非常重要的一部分，它为操作系统识别并挂载U盘、磁盘等提供了保证。

2.2 调试建议：Transport endpoint is not connected以及Input/Output Error

这两个错误已经有不少同学向我们反映。但我们也难以**提供错误具体原因**，建议大家从以下几个角度入手调试：

- **使用F5断点调试**：手动在 terminal 中输入报错的命令，触发断点，定位为什么出错
- **内存溢出、内存越界**：这类问题一般由 `memcpy`、`strcpy` 或是错误的内存访问导致，需要仔细调试（多使用**二分法、排除法**，例如注释掉会发生什么事）
- **返回值正确与否**：查看每个返回值的错误号是否正确，错误号的返回可以参考SFS
- **手动编译，找Warning**：到 build 目录下，手动运行 `make`，看看终端出现的 `warning`，或许对你的调试有帮助

- 钩子是否勾上（针对附加题）：有部分同学钩子函数没有钩到 operations 中，导致一些奇怪的 BUG



```
21 static struct fuse_operations operations = {
22     .init = sfs_init,
23     .destroy = sfs_destroy,
24     .mkdir = sfs_mkdir,
25     .getattr = sfs_getattr,
26     .readdir = sfs_readdir,
27     .mknod = sfs_mknod,
28     .write = sfs_write,
29     .read = sfs_read,
30     .utimens = sfs_utimens,
31     .truncate = sfs_truncate,
32     .unlink = sfs_unlink,
33     .rmdir = sfs_rmdir,
34     .rename = sfs_rename,
35     .readlink = sfs_readlink,
36     .symlink = sfs_symlink,
37
38     .open = sfs_open,
39     .opendir = sfs_opendir,
40     .access = sfs_access
41 };
42
43 /* SECTION: Function Implementation */
44
45 void* sfs_init(struct fuse_conn_info * conn_info) {
46     if (sfs_mount(sfs_options) != SFS_ERROR_NONE) {
47         SFS_DBG("[%s] mount error\n", __func__);
48         fuse_exit(fuse_get_context()->fuse);
49         return NULL;
50     }
```

- 函数是否包含在了.h中：请自行检查
-