关于OS-lab实验五的补充说明

> 首先表达F-tutorial小组的歉意，对同学们造成了不小程度的困惑。这些错误来自于测试的不完备、考虑的不周到，此外，还有一些不细致。不过考虑到Lab-5生命周期仅有两年（今年是第二次投入使用，更新了更加方便的自动化测试脚本以及介质检测工具），还请同学们多多包涵。接下来我们澄清并修改以下几个错误，这两个BUG主要是由**同学A**与**同学B**（就不透露姓名了）提出，感谢他们

关于OS-lab实验五的补充说明

1. 任务一：测试脚本compile_and_run.sh与指令执行不符导致输出`compile_and_run.sh: line 11: ./ddriver_tests No such file ...`
    - 1.1 问题描述
    - 1.2 解决方案
        - 1.2.1 对于还未开始实验的同学
        - 1.2.2 对于已经开始实验的同学
    - 1.3 鸣谢
2. 任务二：循环不断的`umount: /home/students/xxxx/user-land-filesystem/...`
    - 2.1 问题描述
    - 2.2 原因及解决方案
    - 2.3 鸣谢
3. 任务二：错误的测试指令执行顺序
    - 3.1 问题描述与解决方案
    - 3.2 鸣谢

1. 任务一：测试脚本compile_and_run.sh与指令执行不符导致输出`compile_and_run.sh: line 11: ./ddriver_tests No such file ...`

1.1 问题描述

在任务一中，我们推荐的编译测试指令顺序如下：

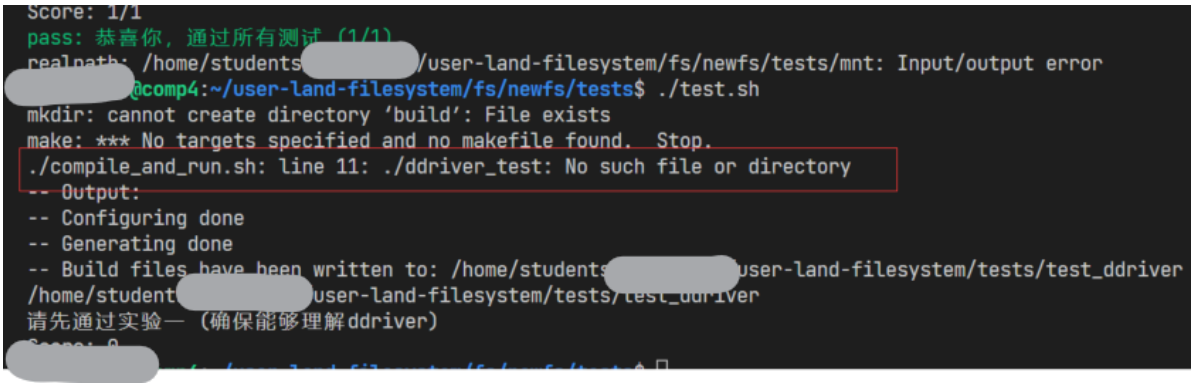
```
cmake CMakeLists.txt
make
./ddriver_test
```

但这会带来多种问题：

1. **目录下文件管理混乱。**例如：有大量`CMake`相关的文件会出现在当前目录下
2. **出现一些未知的BUG。**例如：在执行`tests/main.sh`脚本时，会执行`/tests/test_ddriver`目录下的`compile_and_run.sh`脚本，简化描述如下：

```
# 进入test_ddriver目录
cd ... /user-land-filesystem/tests/test_ddriver
# 创建build文件夹
mkdir build
# 进入build目录
cd build
# cmake ..
cmake ..
# 编译
make
# 执行ddriver_test
./ddriver_test
```

然而，由于第一步`cmake`结果已经在上层目录（即`test_ddriver`目录）生成，导致该步骤`cmake ..`生成的文件不在`build`目录下，而在`test_ddriver`下。我们猜测是`CMakeCache`的问题。这会导致出现如下BUG：



1.2 解决方案

1.2.1 对于还未开始实验的同学

我们已将这几行命令更新为：

```
mkdir build
cd build
cmake ..
make
./ddriver_test
```

这样即可与`compile_and_run.sh`相符，避免后续未知错误。

1.2.2 对于已经开始实验的同学

- **如果测试过程中出现上述BUG**：删掉`test_ddriver`下`cmake`命令生成的一切文件，重新执行上述更新后的命令
- **如果测试过程中未出现上述BUG**：能跑就是赢！

1.3 鸣谢

同学B帮我们指出了这个**BUG**，非常感谢

2. 任务二：循环不断的`umount: /home/students/xxxx/user-land-filesystem/...`

2.1 问题描述

首先感谢两位同学（名字就不透露了）为我们反馈这个重要BUG。该BUG表现如下两图所示（分别来自于**同学A与同学B**）：

2.2 原因及解决方案

- **原因**：问题在于`tests/main.sh`脚本中的`check_mount`函数，该函数用于检查同学们的文件系统是否挂载，代码如下所示：

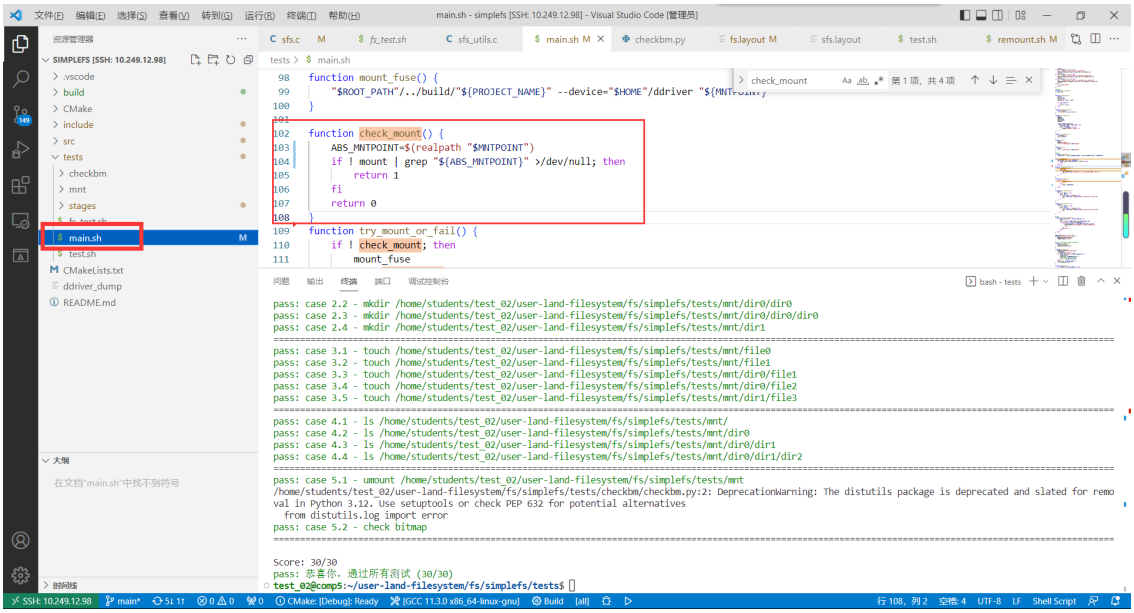
```
function check_mount() {
    if ! mount | grep "${PROJECT_NAME}" >/dev/null; then
        return 1
    fi
    return 0
}
```

这里检查的时候图了方便，仅通过`grep`查`${PROJECT_NAME}`变量有没有在`mount`输出中。然而，这里疏忽了在远程服务器环境下，同学们的`${PROJECT_NAME}`很可能发生冲突，例如都叫`newfs`。因此就算大家正常`umount`了文件系统，**如果此时有别的同学也正挂载着**，那么`check_mount`也会返回1，即卸载失败，那么脚本就会不停地进行`umount`，BUG就产生了。

- **解决方案**：解决方案也很简单，我们判断挂载点路径即可，因为每个同学的挂载点路径一定不同。以SFS为例，大家可以在自己的文件系统工程目录下做如下修改：将`tests`目录下的`main.sh`中大概100行位置的`check_mount`函数修改为以下所示：

```
function check_mount() {
    ABS_MNTPOINT=$(realpath "$MNTPOINT")
    if ! mount | grep "${ABS_MNTPOINT}" >/dev/null; then
        return 1
    fi
    return 0
}
```

其中，`realpath`是一个Linux命令，可以将相对路径变为绝对路径，大家也可以自己在`shell`上敲敲看。那这里`ABS_MNTPOINT`变量就是不同同学挂载点的绝对路径了，OK，BUG解决。



值得说明的是，目前最新的`main.sh`脚本已更新在gitee上：[user-land-filesystem: lab5 filesystem (gitee.com)](), 我们对`template`文件和`simplefs`文件均进行了更新。但是，这需要大家重新生成工程目录文件才能在目录下使用最新的`main.sh`脚本。因此，**我们推荐大家按照上面的方案自行更新，以免造成代码丢失等问题。**

2.3 鸣谢

感谢**同学A**与**同学B**帮忙指出这个BUG，也抱歉浪费大家时间以及对大家造成的巨大困惑。此外，后续**同学A**协助我们进行了脚本正确性测试，再次感谢。

3. 任务二：错误的测试指令执行顺序

3.1 问题描述与解决方案

该问题是F-tutorial团队的细节没有把控好。问题发生在实验指导书提交文档部分，即http://hitsz-cslab.gitee.io/os-labs/lab5/part4/。在最后**remount测试（16分）**一节，原本的指令顺序为：

```
fusermount -u ./tests/mnt
ddriver -r
# F5再次挂载文件系统
mkdir ./tests/mnt/hello
ls ./tests/mnt

python3 ./tests/checkbm/checkbm.py -l ./include/fs.layout -r
./tests/checkbm/golden.json
fusermount -u ./tests/mnt
```

这里我们在还未卸载文件系统时就进行了位图检查（checkbm.py），这显然是不正确的，因为大多数同学的实现是在文件系统卸载后才将内存数据刷回介质，因此正确的指令执行顺序为：

```
fusermount -u ./tests/mnt
ddriver -r
# F5再次挂载文件系统
mkdir ./tests/mnt/hello
ls ./tests/mnt

fusermount -u ./tests/mnt
python3 ./tests/checkbm/checkbm.py -l ./include/fs.layout -r
./tests/checkbm/golden.json
```

该顺序已在指导书上更新，请大家查收。

3.2 鸣谢

最后，该问题同样由**同学A**指出，再次感谢。