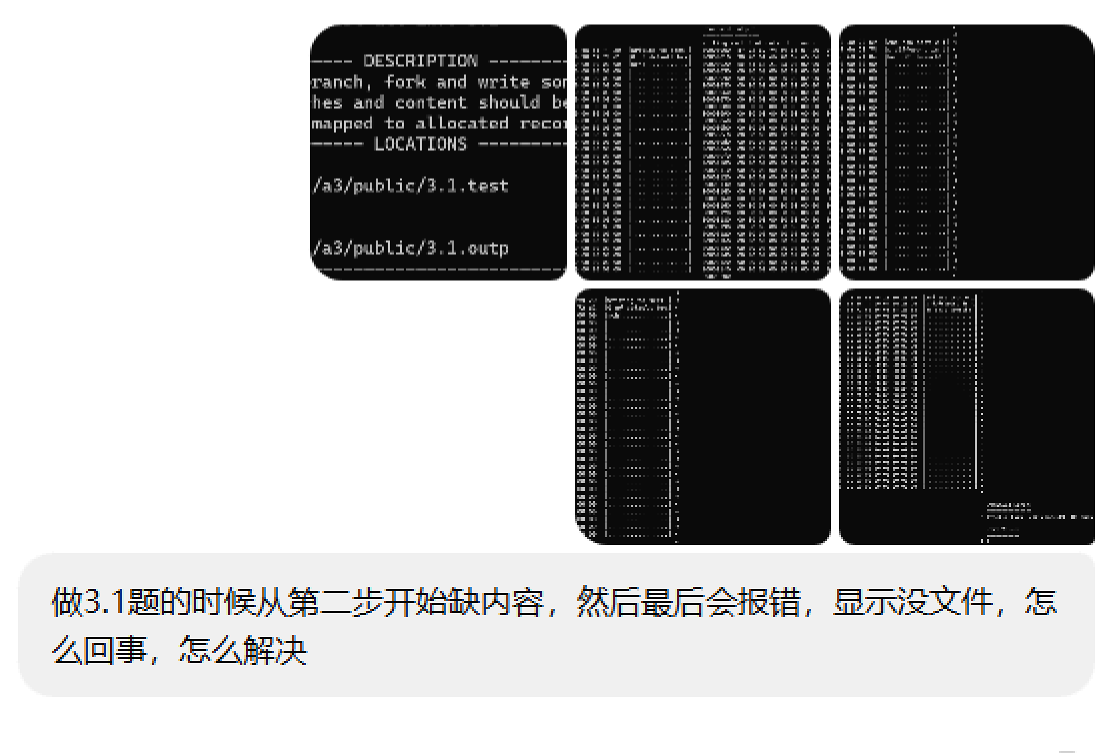
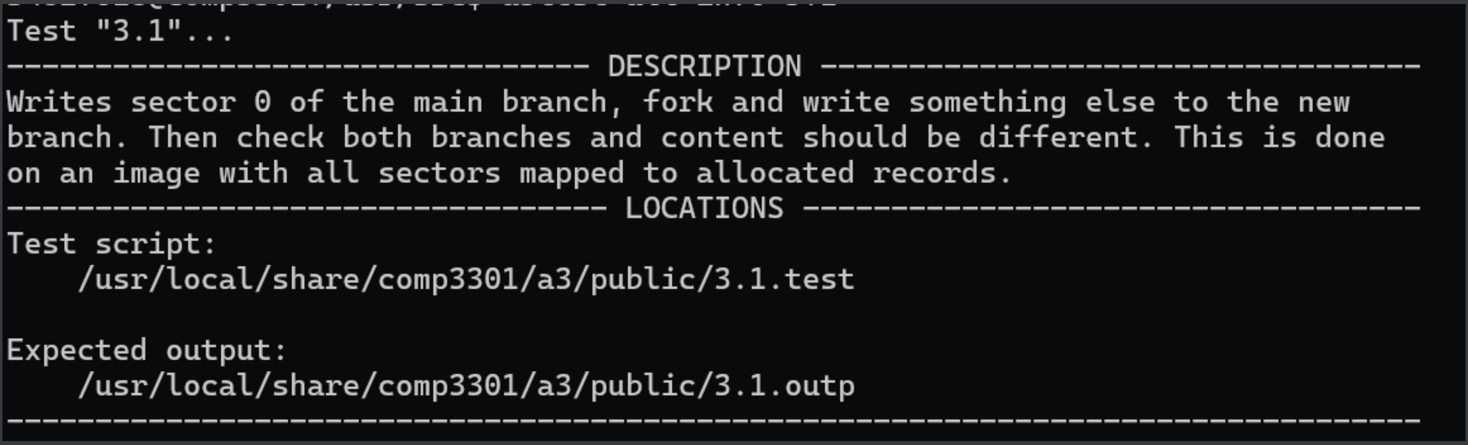
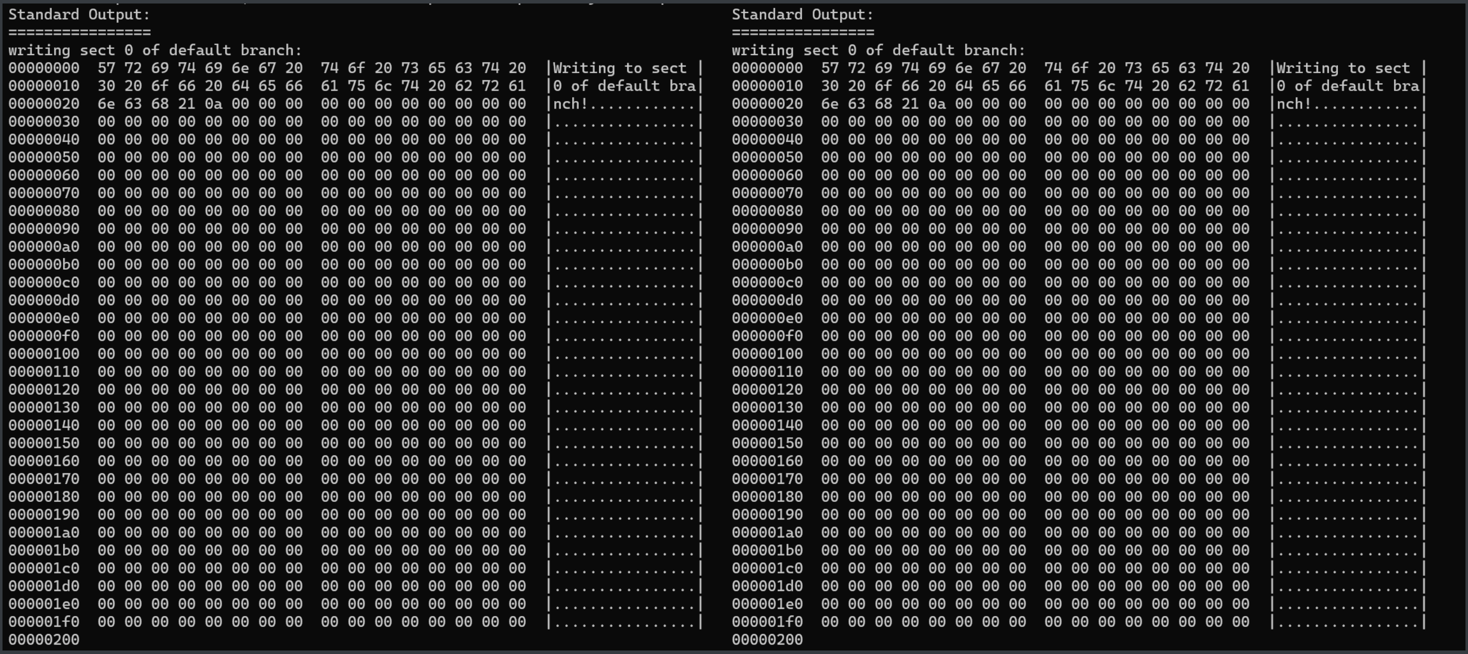


```
---- DESCRIPTION --------
ranch, fork and write so
hes and content should be
mapped to allocated recor
----- LOCATIONS ---------

/a3/public/3.1.test


/a3/public/3.1.outp
```

做3.1题的时候从第二步开始缺内容，然后最后会报错，显示没文件，怎么回事，怎么解决

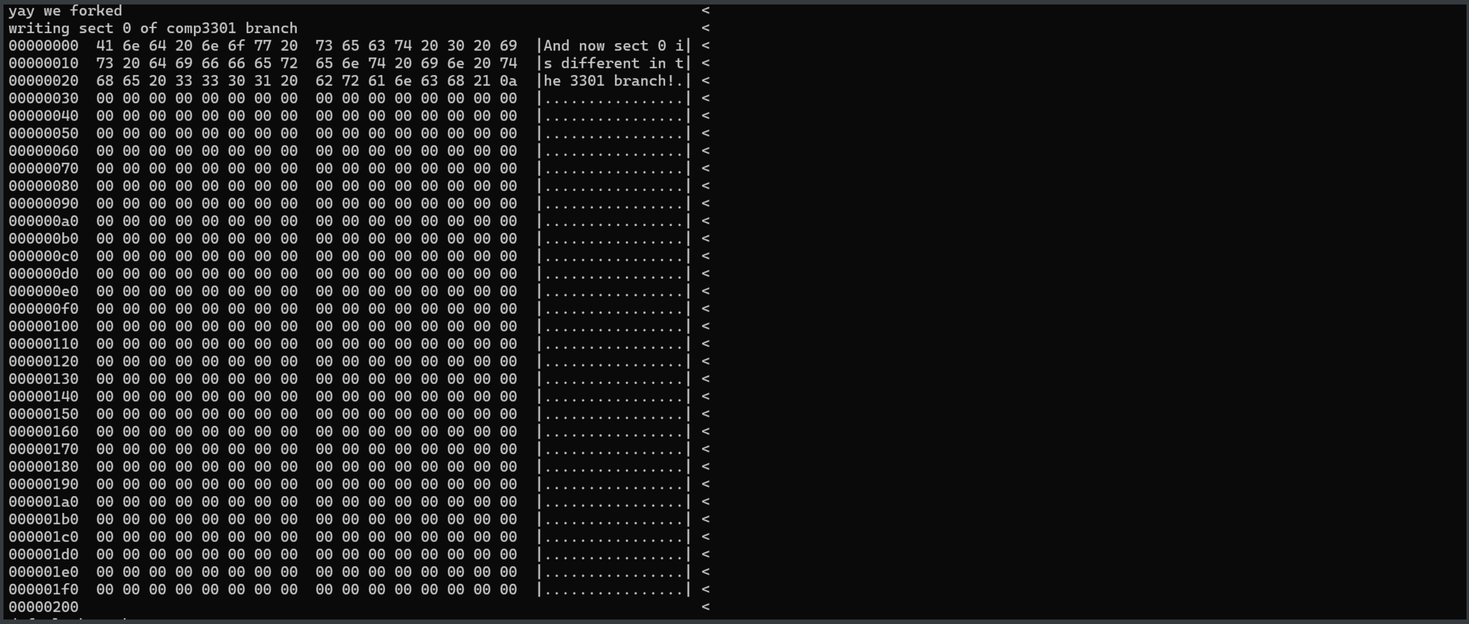


```
Test "3.1"...
---------------------------------- DESCRIPTION ----------------------------------
Writes sector 0 of the main branch, fork and write something else to the new
branch. Then check both branches and content should be different. This is done
on an image with all sectors mapped to allocated records.
----------------------------------- LOCATIONS -----------------------------------
Test script:
    /usr/local/share/comp3301/a3/public/3.1.test

Expected output:
    /usr/local/share/comp3301/a3/public/3.1.outp
---------------------------------------------------------------------------------
```

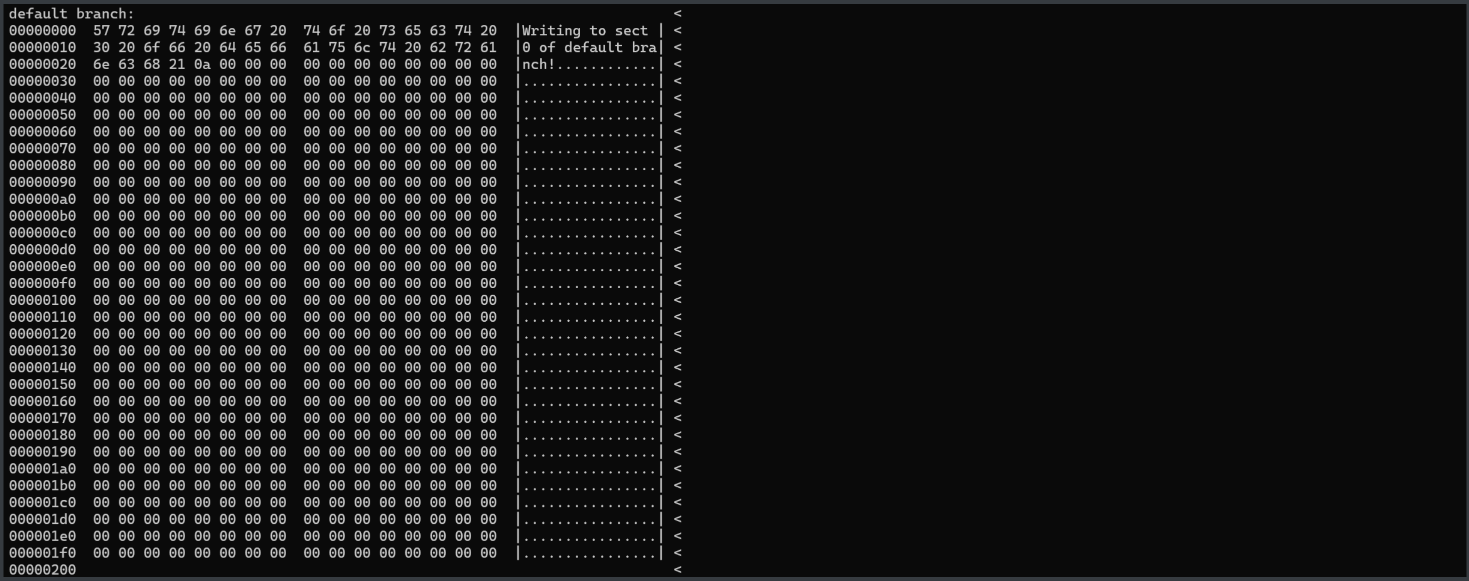


```
Standard Output:
================
writing sect 0 of default branch:
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect |
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra|
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200
```

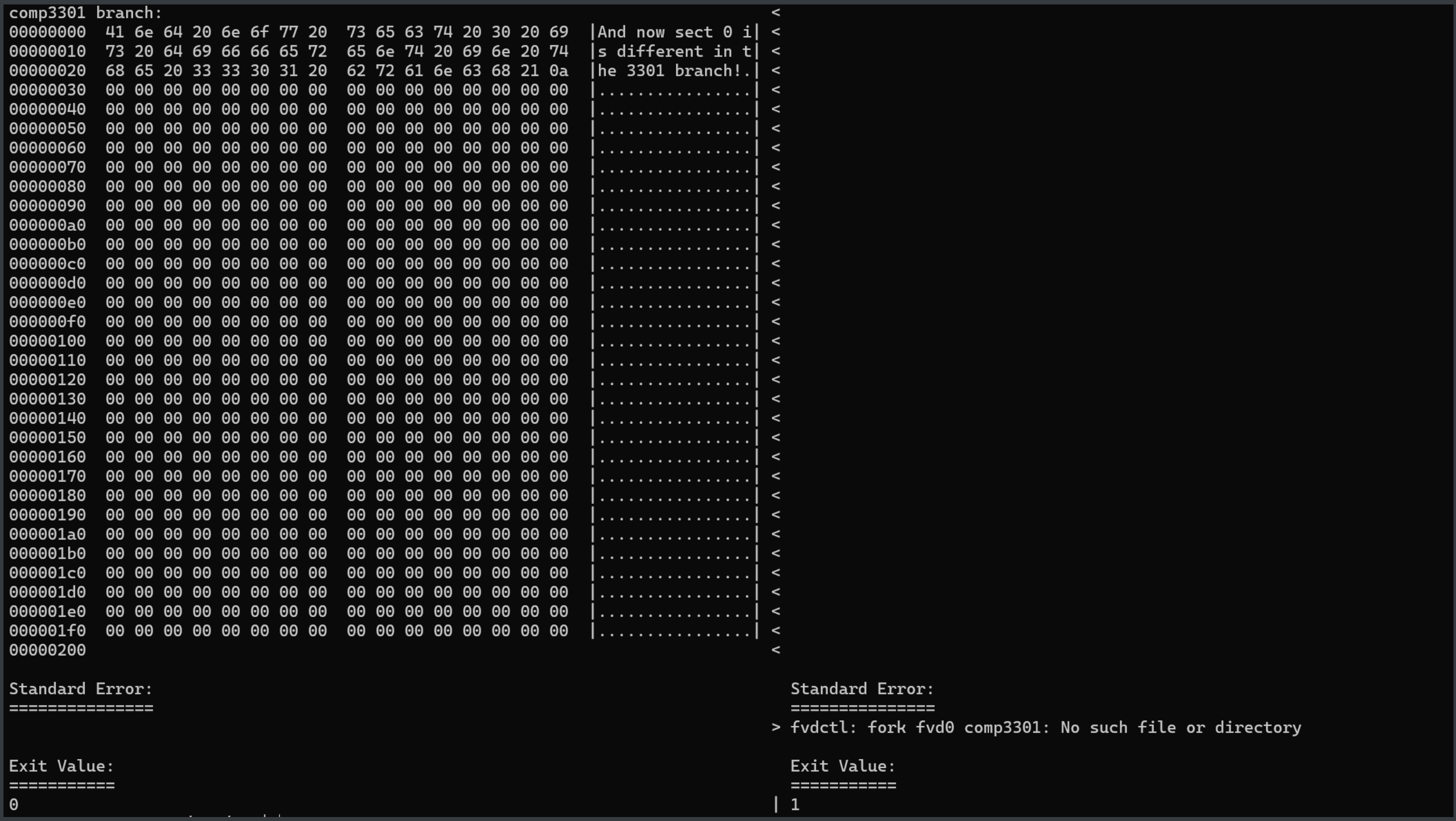
```
Standard Output:
================
writing sect 0 of default branch:
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect |
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra|
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200
```



```
yay we forked                                                                  <
writing sect 0 of comp3301 branch                                              <
00000000  41 6e 64 20 6e 6f 77 20  73 65 63 74 20 30 20 69  |And now sect 0 i| <
00000010  73 20 64 69 66 66 65 72  65 6e 74 20 69 6e 20 74  |s different in t| <
00000020  68 65 20 33 33 30 31 20  62 72 61 6e 63 68 21 0a  |he 3301 branch!.| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <
```



```
default branch:                                                                <
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect | <
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra| <
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <
```



```
comp3301 branch:                                                               <
00000000  41 6e 64 20 6e 6f 77 20  73 65 63 74 20 30 20 69  |And now sect 0 i| <
00000010  73 20 64 69 66 66 65 72  65 6e 74 20 69 6e 20 74  |s different in t| <
00000020  68 65 20 33 33 30 31 20  62 72 61 6e 63 68 21 0a  |he 3301 branch!.| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <

Standard Error:                                                                  Standard Error:
===============                                                                  ===============
                                                                               > fvdctl: fork fvd0 comp3301: No such file or directory

Exit Value:                                                                      Exit Value:
===========                                                                      ===========
0                                                                              | 1
```

根据你提供的图像和问题描述，似乎在做3.1题时，尝试执行某些操作时遇到了文件缺失或错误。报错提示“没有这样的文件或目录”通常意味着程序试图访问一个不存在的文件，可能是由于路径配置不正确或者文件未正确生成。

从图像中可以看到，错误出现在尝试使用 `fvdctl` 工具时，系统找不到所需的文件。

解决这个问题的几个步骤可以是：

1. **确认路径正确**：确保你在命令中使用的路径正确，并且该路径下的文件已经存在。例如，图像中的错误提示“fvdctl: fork fvd0 comp3301: No such file or directory”表明 `comp3301` 文件不存在或路径有误。
2. **检查分支名和文件是否正确生成**：
   - 确保在执行 fork 操作前，所操作的分支（如 `comp3301`）已经正确创建。
   - 确保 `.fvd` 文件和 `.fvd.ref` 文件存在，并且路径配置正确。
3. **验证文件系统状态**：
   - 使用 `fvdtool info` 或 `fvdctl info` 检查当前虚拟磁盘的信息，确保虚拟磁盘文件已经正确挂载。
   - 如果文件丢失，检查你的虚拟磁盘（FVD）创建和挂载过程是否完整。
4. **文件权限问题**：

    - 确保你的程序有访问这些文件的权限，特别是 `.fvd` 和 `.fvd.ref` 文件。你可以使用 `ls -l` 命令检查文件权限。
5. **尝试重新生成或修复缺失的文件**：
    - 如果 `comp3301` 文件丢失，可以尝试重新创建或恢复该文件。

从代码和你提供的描述来看，问题可能出现在以下几个方面，导致无法成功进行 3.1 的测试：

1. 分支创建（Fork）操作未正确实现：

在 `fvd_fork` 操作中，需要确保新分支正确创建并且继承了父分支的数据结构。代码中有执行分支创建的操作，但如果在处理过程中遇到某些错误（如找不到空闲记录、空间不足等），可能导致分支没有成功创建，进而影响测试的执行。

```
case FVDIOC_FORK:
    {
        struct fvd_fork *ff = (struct fvd_fork *)data;  // 获取传入的数据，指向分支创建操作的数据结构
        struct fvd_brch_desc new_branch;  // 新分支的描述符
        uint32_t new_branch_rec;  // 新分支的记录号
        uint32_t i;  // 循环索引，用于遍历块映射记录或其他需要遍历的数组
        uint32_t bmrec, idx;  // bmrec 用于存储块映射记录的索引，idx 用于记录某个特定块映射记录中的条目索引
        uint32_t *table;  // 一个指向 uint32_t 类型数组的指针，用于存储块映射记录的数据
        uint32_t *tab;  // 临时指针，用于在分配内存时保存块映射数据的副本
        uint8_t refval;  // 引用计数值，用于存储某个数据记录的引用计数
        off_t offset;  // 偏移量，表示文件中某个位置的字节偏移，通常用于读取或写入数据
        int error2;  // 用于存储操作的错误码，检测错误并做相应的处理

        uint32_t parent_bmap;  // 父分支的块映射记录
        struct fvd_root_block *root_be;  // 用于根块的操作，最终会写入磁盘

        /* validate new branch name */
        if (ff->ff_name[0] == '\0') {  // 检查分支名称是否为空
            error = EEXIST;  // 如果为空，返回错误：分支名称已存在
            break;
        }

        /* validate branch name length */
        if (strlen(ff->ff_name) >= FVD_MAX_BNM) {  // 检查分支名称长度是否超过最大限制
            error = EINVAL;  // 如果超过最大长度，返回无效的错误
            break;
        }

        /* check if branch already exists */
        error2 = fvd_find_branch(sc, ff->ff_name);  // 检查该分支是否已经存在
        if (error2 == 0) {  // 如果分支已经存在
            error = EEXIST;  // 返回错误：分支已存在
            break;
        }
        if (error2 != ESRCH) {  // 如果查找时发生了其他错误
            error = error2;  // 返回发生的错误
            break;
        }

        /* check if we have space for another branch */
        if (sc->sc_root.fr_nbrches >= FVD_MAX_BRANCHES) {  // 检查是否还有空间来创建新分支sc->sc_root.fr_nbrches：这是 FVD 根块（sc_root）中的一个字段，表示当前 FVD 镜像中已经存在的分支的数量（fr_nbrches）。根块包含了 FVD 镜像的基本信息，包括分支的数量和其他元数据。
            error = ENOSPC;  // 如果没有足够空间，返回错误：没有空间
            break;
        }

        /* check if device is busy */
        if (sc->sc_rw == 0 && ff->ff_force == 0) {  // 检查设备是否处于忙碌状态，且没有强制创建标志
            error = EBUSY;  // 如果设备忙，返回错误：设备忙
            break;
        }

        /* flush all dirty cache entries before fork */
        error = fvd_cache_flush(sc, 0);  // 刷新所有脏缓存，确保数据一致性
        if (error != 0)
            break;  // 如果刷新缓存时出错，退出

        /* allocate new branch descriptor record */
        error = fvd_alloc_record(sc, &new_branch_rec);  // 为新分支分配记录
        if (error != 0)
            break;  // 如果分配记录时出错，退出

        /* check if allocated record is within file bounds */
        if (new_branch_rec >= sc->sc_root.fr_nrecs) {  // 检查新分支记录是否在文件范围内
            error = ENOSPC;  // 如果超出文件范围，返回没有空间的错误
            break;
        }

        /* copy current branch to new branch */
        memcpy(&new_branch, &sc->sc_branch, sizeof(new_branch));  // 复制当前分支的信息到新分支
        new_branch.fb_magic = htobe32(FVD_BRCH_MAGIC);  // 设置分支魔数，确保数据正确性
        uint32_t parent_rec = sc->sc_root.fr_brchs[sc->sc_branch_id];  // 获取父分支的记录号
        new_branch.fb_parent = htobe32(parent_rec);  // 将父分支记录号转换为大端格式
        strlcpy(new_branch.fb_name, ff->ff_name, sizeof(new_branch.fb_name));  // 设置新分支名称

        /* set creation time */
        struct timeval tv;
        microtime(&tv);  // 获取当前时间
        new_branch.fb_ctime = htobe64(tv.tv_sec);  // 设置分支创建时间

        /* allocate and copy block map for child branch */
        uint32_t nrecs_bmap = (sc->sc_root.fr_nsects + FVD_BMAP_ENTRIES - 1) / FVD_BMAP_ENTRIES;  // 计算新分支需要的块映射记录数
        uint32_t first_bmap;  // 记录分配的第一个块映射
        error = fvd_alloc_consecutive(sc, nrecs_bmap, &first_bmap);  // 为新分支分配连续的块映射
        if (error != 0)
            break;  // 如果分配块映射时出错，退出

        /* set refcount to 1 for each new block map record */
        for (uint32_t r = 0; r < nrecs_bmap; r++) {  // 为每个块映射设置引用计数为 1
            error = fvd_ref_write(sc, first_bmap + r, 1);
            if (error != 0)
                break;  // 如果设置引用计数时出错，退出
        }
        if (error != 0)
            break;

        /* copy parent's block map to child's new block map */
        parent_bmap = sc->sc_branch.fb_blkmap;  // 获取父分支的块映射
        tab = malloc(FVD_SECTOR_SIZE, M_TEMP, M_WAITOK);  // 为临时缓冲区分配内存
        for (uint32_t r = 0; r < nrecs_bmap; r++) {  // 将父分支的块映射复制到新分支的块映射
            off_t off_parent = (off_t)(parent_bmap + r) * FVD_SECTOR_SIZE;
            off_t off_child = (off_t)(first_bmap + r) * FVD_SECTOR_SIZE;

            // 读取父分支的块映射记录
            error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)tab,
                FVD_SECTOR_SIZE, off_parent, UIO_SYSSPACE, IO_NODELOCKED,
                sc->sc_ucred, NULL, curproc);
            if (error != 0)
                break;

            // 写入新分支的块映射记录
            error = vn_rdwr(UIO_WRITE, sc->sc_fvdvp, (caddr_t)tab,
                FVD_SECTOR_SIZE, off_child, UIO_SYSSPACE, IO_NODELOCKED,
                sc->sc_ucred, NULL, curproc);
            if (error != 0)
                break;
        }
        free(tab, M_TEMP, FVD_SECTOR_SIZE);  // 释放临时缓冲区
        if (error != 0)
            break;

        /* update new_branch.fb_blkmap to point to new block map */
        new_branch.fb_blkmap = htobe32(first_bmap);  // 更新新分支的块映射指向新分配的块映射

        /* write new branch descriptor */
        offset = (off_t)new_branch_rec * FVD_SECTOR_SIZE;  // 计算新分支描述符的偏移量
        error = vn_rdwr(UIO_WRITE, sc->sc_fvdvp, (caddr_t)&new_branch,
            sizeof(new_branch), offset, UIO_SYSSPACE, IO_NODELOCKED,
            sc->sc_ucred, NULL, curproc);  // 写入新的分支描述符
        if (error != 0)
            break;

        /* increment reference counts for all shared data records */
        table = malloc(FVD_SECTOR_SIZE, M_TEMP, M_WAITOK);  // 为共享的数据记录分配临时表
        for (i = 0; i < sc->sc_root.fr_nsects; i++) {  // 遍历所有数据记录
            bmrec = sc->sc_branch.fb_blkmap + (i / FVD_BMAP_ENTRIES);
            idx = i % FVD_BMAP_ENTRIES;

            // 读取块映射记录
            offset = (off_t)bmrec * FVD_SECTOR_SIZE;
            error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)table,
                FVD_SECTOR_SIZE, offset, UIO_SYSSPACE, IO_NODELOCKED,
                sc->sc_ucred, NULL, curproc);
            if (error != 0)
                break;

            // 检查是否有数据记录，并增加引用计数
            uint32_t recno = betoh32(((uint32_t *)table)[idx]);
            if (recno != 0) {
                error = fvd_ref_read(sc, recno, &refval);  // 读取引用计数
                if (error != 0)
                    break;
                refval++;  // 增加引用计数
                error = fvd_ref_write(sc, recno, refval);  // 写回引用计数
                if (error != 0)
                    break;
            }
        }
        free(table, M_TEMP, FVD_SECTOR_SIZE);  // 释放临时表
        if (error != 0)
            break;

        /* update root block in host order */
        uint16_t new_nbrches = sc->sc_root.fr_nbrches + 1;  // 更新根块中的分支计数
        sc->sc_root.fr_nbrches = new_nbrches;  // 保持主机字节顺序
        sc->sc_root.fr_brchs[new_nbrches - 1] = new_branch_rec;  // 保存新分支记录
        /* create big-endian copy for writing */
        root_be = malloc(sizeof(struct fvd_root_block), M_TEMP, M_WAITOK);  // 创建根块的副本
        *root_be = sc->sc_root;
        root_be->fr_magic = htobe32(FVD_ROOT_BLK_MAGIC);  // 转换为大端格式
        root_be->fr_vmaj = FVD_VER_MAJ;
        root_be->fr_vmin = FVD_VER_MIN;
        root_be->fr_nbrches = htobe16(sc->sc_root.fr_nbrches);
        root_be->fr_nrecs = htobe32(sc->sc_root.fr_nrecs);
        root_be->fr_nsects = htobe32(sc->sc_root.fr_nsects);
        root_be->fr_ncyls = htobe32(sc->sc_root.fr_ncyls);
        root_be->fr_nheads = htobe16(sc->sc_root.fr_nheads);
        root_be->fr_nspt = htobe16(sc->sc_root.fr_nspt);

        // 将分支数组转换为大端格式
        for (uint16_t i = 0; i < sc->sc_root.fr_nbrches; i++) {
            root_be->fr_brchs[i] = htobe32(sc->sc_root.fr_brchs[i]);
        }

        /* write updated root block */
        error = vn_rdwr(UIO_WRITE, sc->sc_fvdvp, (caddr_t)root_be,
            FVD_SECTOR_SIZE, 0, UIO_SYSSPACE, IO_NODELOCKED,
            sc->sc_ucred, NULL, curproc);  // 将更新后的根块写回
        free(root_be, M_TEMP, sizeof(*root_be));  // 释放根块副本
        if (error != 0)
            break;

        /* switch to new branch */
        memcpy(&sc->sc_branch, &new_branch, sizeof(sc->sc_branch));  // 切换到新分支
        sc->sc_branch_id = new_nbrches - 1;  // 使用新分支的索引
        strlcpy(sc->sc_bname, ff->ff_name, sizeof(sc->sc_bname));  // 设置新分支的名称

        /* clear cache since we switched branches */
        fvd_cache_flush(sc, 1);  // 清除缓存，因为我们已经切换了分支
    }
    break;
```

2. 拷贝操作（Copy-on-write）问题：

在分支创建过程中，涉及到拷贝操作（COW）。`fvd_needs_cow` 和 `fvd_do_cow` 的实现主要用于处理分支间的写入操作时，确保父分支的数据不会被覆盖。可能是这部分的拷贝操作未正确完成，导致分支内容无法与父分支区分开。

```
/*
 * Check if sector needs copy-on-write
 */
static int
fvd_needs_cow(struct fvd_softc *sc, uint32_t sec)  // 定义一个函数，检查给定的扇区(sector) 是否需要进行拷贝写操作（Copy-on-Write，COW）
{
    uint32_t recno = 0;  // 定义一个变量 recno 来存储与该扇区相关的记录号
    uint8_t refval;  // 定义一个变量 refval 用来存储该记录的引用计数（reference count）
    int error;  // 用来存储函数调用时的错误码

    error = fvd_bmap_get(sc, sec, &recno);  // 调用 fvd_bmap_get 函数，通过给定的扇区号获取该扇区的记录号 (recno)
    if (error != 0 || recno == 0)  // 如果获取记录号时发生错误 (error != 0)，或者该扇区未被分配（recno == 0）
        return 0;  // 返回 0，表示不需要拷贝写操作，因为该扇区为空或无效

    error = fvd_ref_read(sc, recno, &refval);  // 调用 fvd_ref_read 函数，读取与该记录相关的引用计数 (refval)
    if (error != 0)  // 如果读取引用计数时发生错误
        return 0;  // 返回 0，表示不需要进行拷贝写操作

    return (refval > 1);  // 如果该记录的引用计数大于 1，表示该记录被多个分支引用，返回 1，表示需要拷贝写操作
                          // 否则，返回 0，表示不需要拷贝写操作
}
```

```
/*
```

```
 * Perform copy-on-write for sector
 */
static int
fvd_do_cow(struct fvd_softc *sc, uint32_t sec)  // 定义一个函数，执行拷贝写操作（Copy-on-Write，COW），针对指定的扇区 `sec`
{
    uint32_t oldrec, newrec;  // oldrec 存储旧的记录号，newrec 存储新的记录号
    uint8_t refval;  // 用于存储读取到的引用计数值
    uint8_t buf[FVD_SECTOR_SIZE];  // 缓冲区，用于存储要复制的扇区数据
    int error;  // 用于存储函数调用过程中出现的错误码

    /* read old mapping */
    error = fvd_bmap_get(sc, sec, &oldrec);  // 通过扇区 `sec` 获取该扇区的记录号 `oldrec`
    if (error != 0 || oldrec == 0)  // 如果获取记录号失败，或者该扇区未分配（即 `oldrec == 0`）
        return 0;  // 返回 0，表示不需要进行拷贝写操作

    /* decrement old record refcount */
    error = fvd_ref_read(sc, oldrec, &refval);  // 读取当前记录号 `oldrec` 的引用计数 `refval`
    if (error != 0)  // 如果读取引用计数时发生错误
        return error;  // 返回错误码

    if (refval > 0) {  // 如果引用计数大于 0，表示该记录有其他分支引用
        refval--;  // 将引用计数减 1
        error = fvd_ref_write(sc, oldrec, refval);  // 更新 `oldrec` 的引用计数
        if (error != 0)  // 如果更新引用计数时发生错误
            return error;  // 返回错误码
    }

    /* allocate new record */
    error = fvd_alloc_record(sc, &newrec);  // 为新记录分配一个新的记录号 `newrec`
    if (error != 0)  // 如果分配新记录失败
        return error;  // 返回错误码

    /* copy old data */
    error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)buf, FVD_SECTOR_SIZE,  // 从磁盘读取 `oldrec` 记录的数据到缓冲区 `buf`
        (off_t)oldrec * FVD_SECTOR_SIZE, UIO_SYSSPACE,
        IO_NODELOCKED, sc->sc_ucred, NULL, curproc);
    if (error != 0)  // 如果读取数据时出错
        return error;  // 返回错误码

    /* write to new record */
    error = vn_rdwr(UIO_WRITE, sc->sc_fvdvp, (caddr_t)buf, FVD_SECTOR_SIZE,  // 将数据从缓冲区 `buf` 写入到新的记录 `newrec`
        (off_t)newrec * FVD_SECTOR_SIZE, UIO_SYSSPACE,
        IO_NODELOCKED, sc->sc_ucred, NULL, curproc);
    if (error != 0)  // 如果写入数据时出错
        return error;  // 返回错误码

    /* update mapping to point to new record */
    error = fvd_bmap_set(sc, sec, newrec);  // 更新块映射，将该扇区 `sec` 映射到新的记录号 `newrec`
    if (error != 0)  // 如果更新映射时出错
        return error;  // 返回错误码

    /* set new record refcount = 1 */
    refval = 1;  // 为新记录设置引用计数为 1
    error = fvd_ref_write(sc, newrec, refval);  // 将引用计数写回新记录 `newrec`
    if (error != 0)  // 如果写入引用计数时出错
        return error;  // 返回错误码

    return 0;  // 成功完成拷贝写操作，返回 0 表示没有错误
}
```

3. 文件路径或挂载问题:

在尝试 fork 操作时，可能是文件路径的配置或挂载的问题，导致测试脚本无法找到期望的 `.fvd` 或 `.fvd.ref` 文件。需要确认文件路径、分支文件及元数据文件（如 `.fvd` ）是否在正确的位置，并且已经挂载。

4. 缓存和数据一致性：

代码中的 `fvd_cache_add` 、 `fvd_cache_flush` 处理了数据的缓存和写回，确保修改后的数据正确写入磁盘。如果缓存没有正确刷新或者未正确更新，可能导致分支内容无法显示为预期。

```
/*
 * Add sector to cache using LRU replacement
 */
static void
fvd_cache_add(struct fvd_softc *sc, uint32_t sec, const void *buf,  // 定义一个函数，将给定的扇区数据 `buf` 添加到缓存中，使用LRU（最近最少使用）替换策略
    uint32_t checksum, int dirty)  // 参数分别是扇区号 `sec`，数据缓冲区 `buf`，数据的校验和 `checksum` 和数据的脏标志 `dirty`
{
    struct fvd_cache_entry *entry, *lru_entry;  // `entry` 用于表示缓存中的一个缓存条目，`lru_entry` 用于表示被选择为最近最少使用（LRU）的缓存条目
    uint64_t oldest_time;  // `oldest_time` 用于记录最近最少使用条目的时间戳
    int i;  // 循环计数器，用于遍历缓存中的条目

    /* If the sector already exists in cache, update in place. */
    rw_enter_write(&sc->sc_cache_lock);  // 获取缓存锁，以确保对缓存的写操作是线程安全的
    entry = fvd_cache_find(sc, sec);  // 查找是否有对应扇区的缓存条目
    if (entry != NULL) {  // 如果该扇区已经存在于缓存中
        memcpy(entry->fce_data, buf, FVD_SECTOR_SIZE);  // 更新缓存条目的数据
        entry->fce_checksum = checksum;  // 更新缓存条目的校验和
        entry->fce_dirty = dirty ? 1 : entry->fce_dirty;  // 如果数据是脏的，则标记为脏，否则保持原值
        entry->fce_valid = 1;  // 标记缓存条目为有效
        entry->fce_access_count = ++sc->sc_access_counter;  // 增加访问计数，用于LRU策略
        rw_exit_write(&sc->sc_cache_lock);  // 释放缓存锁
        return;  // 返回，说明该扇区数据已更新，无需进行进一步操作
    }

    /* Select victim with LRU (or free slot). */
    lru_entry = NULL;  // 初始化 `lru_entry` 为 NULL
    oldest_time = UINT64_MAX;  // 初始化最老的时间戳为最大值
    for (i = 0; i < 16; i++) {  // 遍历缓存中的每个条目（假设缓存最多有 16 个条目）
        entry = &sc->sc_cache[i];  // 获取当前缓存条目
```

```c
        if (!entry->fce_valid) {  // 如果当前条目无效（即空闲条目）
            lru_entry = entry;  // 选择当前条目作为新的空闲条目
            break;  // 跳出循环，因为找到一个空闲条目
        }
        if (entry->fce_access_count < oldest_time) {  // 如果当前条目的访问计数小于最老的时间戳
            oldest_time = entry->fce_access_count;  // 更新最老的时间戳
            lru_entry = entry;  // 选择当前条目作为最近最少使用条目
        }
    }

    /* Snapshot victim (if any) for writeback, then drop lock during I/O. */
    uint8_t victim_data[FVD_SECTOR_SIZE];  // 为被替换的条目分配一个缓冲区 `victim_data` 用于存储数据
    uint32_t victim_sector = 0;  // 被替换的扇区号
    int need_writeback = 0;  // 是否需要将被替换的条目写回磁盘

    if (lru_entry != NULL && lru_entry->fce_valid && lru_entry->fce_dirty) {  // 如果被选择的条目有效且是脏的（即已修改）
        memcpy(victim_data, lru_entry->fce_data, sizeof(victim_data));  // 保存当前条目的数据到 `victim_data`
        victim_sector = lru_entry->fce_sector;  // 保存被替换的扇区号
        need_writeback = 1;  // 设置需要写回磁盘
    }
    rw_exit_write(&sc->sc_cache_lock);  // 在执行 I/O 操作之前释放缓存锁

    /* Writeback outside the lock. */
    if (need_writeback) {  // 如果需要将被替换的条目写回磁盘
        /* create a temporary cache entry for writeback */
        struct fvd_cache_entry temp_entry;  // 创建一个临时的缓存条目 `temp_entry`
        temp_entry.fce_sector = victim_sector;  // 设置该条目的扇区号
        temp_entry.fce_dirty = 1;  // 设置该条目为脏
        temp_entry.fce_valid = 1;  // 设置该条目为有效
        memcpy(temp_entry.fce_data, victim_data, FVD_SECTOR_SIZE);  // 将数据拷贝到临时条目中
        (void)fvd_cache_writeback(sc, &temp_entry);  // 将该临时条目写回磁盘
    }
    /* Finally install the new entry. */
    rw_enter_write(&sc->sc_cache_lock);  // 重新获取缓存锁
    /* lru_entry can’t be NULL since cache has fixed 16 entries. */
    lru_entry->fce_sector = sec;  // 设置新条目的扇区号
    lru_entry->fce_checksum = checksum;  // 设置新条目的校验和
    lru_entry->fce_dirty = dirty ? 1 : 0;  // 设置新条目的脏标志
    lru_entry->fce_valid = 1;  // 设置新条目为有效
    lru_entry->fce_access_count = ++sc->sc_access_counter;  // 增加访问计数
    memcpy(lru_entry->fce_data, buf, FVD_SECTOR_SIZE);  // 将数据拷贝到新条目中
    rw_exit_write(&sc->sc_cache_lock);  // 释放缓存锁
}
```

```
/*
 * Flush all dirty cache entries. If `invalidate` is non-zero, also invalidate
 * every entry afterwards (used by detach and CACHE_EMPTY).
 */
static int
fvd_cache_flush(struct fvd_softc *sc, int invalidate)  // 定义一个函数，用于刷新所有脏缓存条目。如果 `invalidate` 非零，还会使缓存条目失效
{
    int i, error = 0, oerror = 0;  // 定义变量 `i` 为循环计数器，`error` 和 `oerror` 用于存储错误代码
    /*
     * We don't hold sc_cache_lock while doing vn_rdwr() I/O to avoid
     * blocking other code paths which might also want the lock. We
     * snapshot the flush plan under the lock first.
     */
    struct {  // 定义一个结构体数组 `plan` 用于记录待刷新的缓存条目的状态
        int used;  // 标记条目是否需要刷新（脏的）
        uint32_t sector;  // 缓存条目的扇区号
        int idx;  // 缓存条目在数组中的索引
    } plan[16];

    rw_enter_read(&sc->sc_cache_lock);  // 获取缓存锁，确保对缓存的读取是线程安全的
    for (i = 0; i < 16; i++) {  // 遍历缓存中的每个条目（假设最多有 16 个缓存条目）
        plan[i].used = (sc->sc_cache[i].fce_valid && sc->sc_cache[i].fce_dirty);  // 如果条目有效且脏，标记为待刷新
        plan[i].sector = sc->sc_cache[i].fce_sector;  // 保存缓存条目的扇区号
        plan[i].idx = i;  // 保存缓存条目的索引
    }
    rw_exit_read(&sc->sc_cache_lock);  // 释放缓存锁，因为我们已经完成了缓存状态的读取

    for (i = 0; i < 16; i++) {  // 遍历缓存条目计划 `plan` 数组
        if (!plan[i].used)  // 如果当前条目不需要刷新，跳过
            continue;

        /* Re-acquire write lock only to fetch pointer safely. */
        rw_enter_write(&sc->sc_cache_lock);  // 获取写锁，以便安全地访问缓存条目
        struct fvd_cache_entry *e = &sc->sc_cache[plan[i].idx];  // 获取待刷新条目的指针
        /* Sector might have changed; re-check. */
        if (!(e->fce_valid && e->fce_dirty && e->fce_sector == plan[i].sector)) {  // 如果条目已经无效或被其他线程修改
            rw_exit_write(&sc->sc_cache_lock);  // 释放锁，跳过该条目
            continue;
        }
        /* Make a local copy of the data then drop the lock for I/O. */
        uint8_t local[FVD_SECTOR_SIZE];  // 定义一个临时缓冲区 `local` 用于存储条目的数据
        memcpy(local, e->fce_data, sizeof(local));  // 将条目数据复制到 `local` 缓冲区
        uint32_t sec = e->fce_sector;  // 获取当前条目的扇区号
        rw_exit_write(&sc->sc_cache_lock);  // 释放缓存锁，准备进行 I/O 操作

        /* Perform the actual writeback without holding the lock. */
        struct fvd_cache_entry temp_entry;  // 创建一个临时缓存条目 `temp_entry`
        temp_entry.fce_sector = sec;  // 设置扇区号
        temp_entry.fce_dirty = 1;  // 设置为脏标记
        temp_entry.fce_valid = 1;  // 设置为有效
        memcpy(temp_entry.fce_data, local, FVD_SECTOR_SIZE);  // 将数据拷贝到临时条目
        error = fvd_cache_writeback(sc, &temp_entry);  // 将临时条目写回磁盘
        if (error != 0 && oerror == 0)  // 如果写回过程中发生错误，并且 `oerror` 还未设置
            oerror = error;  // 将错误码存入 `oerror` 以供后续处理

        /* Mark clean / invalidate under lock. */
        rw_enter_write(&sc->sc_cache_lock);  // 重新获取写锁，准备标记缓存条目
        e = &sc->sc_cache[plan[i].idx];  // 获取当前条目
        if (e->fce_valid && e->fce_sector == sec) {  // 如果该条目仍然有效并且未被替换
```

```
            if (error == 0) {  // 如果没有发生写回错误
                e->fce_dirty = 0;  // 将条目标记为干净
                e->fce_checksum = fvd_csum_sect(local);  // 重新计算校验和
            }
            if (invalidate) {  // 如果需要失效操作
                e->fce_valid = 0;  // 将条目标记为无效
            }
        }
        rw_exit_write(&sc->sc_cache_lock);  // 释放缓存锁
    }

    if (invalidate) {  // 如果 `invalidate` 非零，表示需要使所有缓存条目失效
        rw_enter_write(&sc->sc_cache_lock);  // 获取写锁
        /* Invalidate all cache entries */
        for (i = 0; i < 16; i++) {  // 遍历缓存中的所有条目
            sc->sc_cache[i].fce_valid = 0;  // 将所有条目标记为无效
        }
        sc->sc_cache_size = 0;  // 清空缓存的大小
        rw_exit_write(&sc->sc_cache_lock);  // 释放缓存锁
    }
    return oerror;  // 返回最后发生的错误码，如果没有错误则返回 0
}
```

5. 元数据读取问题：

`fvd_read_metadata` 和 `fvd_find_branch` 函数负责加载和验证磁盘镜像的元数据。如果元数据读取失败（例如文件结构损坏、分支名错误等），也会导致测试无法继续执行。

```
/*
 * Read FVD metadata from file
 */
static int
fvd_read_metadata(struct fvd_softc *sc)  // 定义一个函数，从文件中读取 FVD（虚拟分支磁盘）元数据，并存储到 `sc` 对象中
{
```

```c
    int error;  // 用于存储函数调用的错误码
    off_t offset;  // 用于表示读取操作的偏移量

    /* read root block */
    offset = 0;  // 根块（`root block`）从文件的开始位置读取
    error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)&sc->sc_root,  // 使用 `vn_rdwr` 函数从文件中读取根块数据
        sizeof(sc->sc_root), offset, UIO_SYSSPACE, IO_NODELOCKED,  // 读取 `sc_root` 大小的字节数
        sc->sc_ucred, NULL, curproc);  // 使用用户凭证进行读取
    if (error != 0)  // 如果读取操作出错
        return (error);  // 返回错误码

    /* validate magic */
    if (betoh32(sc->sc_root.fr_magic) != FVD_ROOT_BLK_MAGIC)  // 验证根块的 magic 值是否与预期的 `FVD_ROOT_BLK_MAGIC` 匹配
        return (EINVAL);  // 如果不匹配，返回无效参数错误

    /* Return EINVAL for invalid FVD image as per 'appropriate errno(2)' */

    if (sc->sc_root.fr_vmaj != FVD_VER_MAJ ||  // 检查根块中的版本号是否与预期的版本号匹配
        sc->sc_root.fr_vmin != FVD_VER_MIN)  // 检查次版本号是否匹配
        return (EINVAL);  // 如果版本号不匹配，返回无效参数错误

    /* convert big-endian fields to host byte order */
    sc->sc_root.fr_nbrches = betoh16(sc->sc_root.fr_nbrches);  // 将根块中的大端格式的字段转换为主机字节序（如：分支数量）
    sc->sc_root.fr_nrecs = betoh32(sc->sc_root.fr_nrecs);  // 将大端格式的记录数转换为主机字节序
    sc->sc_root.fr_nsects = betoh32(sc->sc_root.fr_nsects);  // 将大端格式的扇区数转换为主机字节序
    sc->sc_root.fr_ncyls = betoh32(sc->sc_root.fr_ncyls);  // 将大端格式的柱面数转换为主机字节序
    sc->sc_root.fr_nheads = betoh16(sc->sc_root.fr_nheads);  // 将大端格式的盘头数转换为主机字节序
    sc->sc_root.fr_nspt = betoh16(sc->sc_root.fr_nspt);  // 将大端格式的每磁道扇区数转换为主机字节序
    /* convert branch array */
    for (uint32_t i = 0; i < sc->sc_root.fr_nbrches && i < nitems(sc->sc_root.fr_brchs); i++) {
        sc->sc_root.fr_brchs[i] = betoh32(sc->sc_root.fr_brchs[i]);  // 将根块中的分支记录数组 `fr_brchs` 从大端格式转换为主机字节序
    }

    /* find and read branch descriptor */
    error = fvd_find_branch(sc, sc->sc_bname);  // 调用 `fvd_find_branch` 函数查找并读取指定分支的描述符
    if (error != 0)  // 如果查找分支时出错
        return (error);  // 返回错误码

    /* set disk geometry from host-order fields */
    sc->sc_cylinders = sc->sc_root.fr_ncyls;  // 将根块中的柱面数赋值给 `sc_cylinders`
    sc->sc_heads = sc->sc_root.fr_nheads;  // 将根块中的盘头数赋值给 `sc_heads`
    sc->sc_spt = sc->sc_root.fr_nspt;  // 将根块中的每磁道扇区数赋值给 `sc_spt`

    return (0);  // 成功读取元数据并完成所有操作，返回 0
}
```

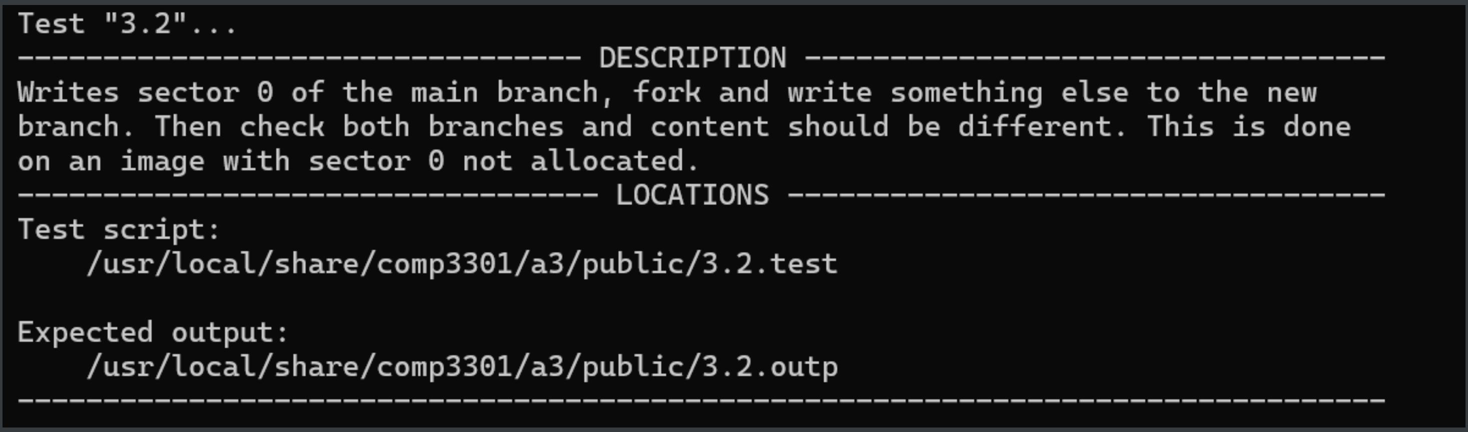
```
/*
 * Find branch descriptor by name
 */
static int
fvd_find_branch(struct fvd_softc *sc, const char *bname)  // 定义一个函数，通过分支名查找分支描述符
{
    struct fvd_brch_desc brch;  // 定义一个结构体 `brch` 用于存储分支描述符
    off_t offset;  // 偏移量，用于读取分支描述符的具体位置
    int error;  // 存储函数调用的错误码
    uint32_t i;  // 循环变量，用于遍历分支描述符

    if (bname[0] == '\0') {  // 如果分支名为空字符串
        offset = (off_t)sc->sc_root.fr_brchs[0] * FVD_SECTOR_SIZE;  // 获取第一个分支描述符的偏移量
        error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)&brch, sizeof(brch),  // 从文件读取分支描述符到 `brch`
            offset, UIO_SYSSPACE, IO_NODELOCKED, sc->sc_ucred,
            NULL, curproc);
        if (error != 0)  // 如果读取过程中发生错误
            return (error);  // 返回错误码

        /* sanity check and endian fixups for branch descriptor */
        if (betoh32(brch.fb_magic) != FVD_BRCH_MAGIC)  // 检查分支描述符的 magic 值是否匹配
            return (EINVAL);  // 如果不匹配，返回无效参数错误

        /* convert big-endian fields to host byte order */
        uint16_t nchilds = betoh16(brch.fb_nchilds);  // 将大端格式的字段转换为主机字节序
        uint64_t ctime = betoh64(brch.fb_ctime);
        uint32_t blkmap = betoh32(brch.fb_blkmap);
        uint32_t parent = betoh32(brch.fb_parent);
        for (int k = 0; k < 16; k++)  // 将 `fb_child` 数组中的大端字段转换为主机字节序
            brch.fb_child[k] = betoh32(brch.fb_child[k]);

        /* cache back to host-order copy */
        brch.fb_magic = FVD_BRCH_MAGIC;  // 将字段转换回主机字节序
        brch.fb_nchilds = nchilds;
        brch.fb_ctime = ctime;
        brch.fb_blkmap = blkmap;
        brch.fb_parent = parent;

        sc->sc_branch = brch;  // 将当前读取的分支描述符存储到 `sc->sc_branch`
        sc->sc_branch_id = 0;  // 设置当前分支 ID 为 0
        return (0);  // 返回 0，表示找到分支
    }

    /* search through branch descriptors */
    for (i = 0; i < sc->sc_root.fr_nbrches; i++) {  // 遍历所有的分支描述符
        offset = (off_t)sc->sc_root.fr_brchs[i] * FVD_SECTOR_SIZE;  // 获取当前分支描述符的偏移量
        error = vn_rdwr(UIO_READ, sc->sc_fvdvp, (caddr_t)&brch,  // 读取分支描述符数据
            sizeof(brch), offset, UIO_SYSSPACE, IO_NODELOCKED,
```

```c
            sc->sc_ucred, NULL, curproc);
        if (error != 0)  // 如果读取过程中发生错误
            return (error);  // 返回错误码

        /* sanity check and endian fixups for branch descriptor */
        if (betoh32(brch.fb_magic) != FVD_BRCH_MAGIC)  // 检查分支描述符的 magic 值
            continue;  // 如果 magic 值不匹配，跳过当前分支

        /* convert big-endian fields to host byte order */
        uint16_t nchilds = betoh16(brch.fb_nchilds);  // 将大端格式字段转换为主机字节序
        uint64_t ctime = betoh64(brch.fb_ctime);
        uint32_t blkmap = betoh32(brch.fb_blkmap);
        uint32_t parent = betoh32(brch.fb_parent);
        for (int k = 0; k < 16; k++)  // 将 `fb_child` 数组中的大端字段转换为主机字节序
            brch.fb_child[k] = betoh32(brch.fb_child[k]);

        /* cache back to host-order copy */
        brch.fb_magic = FVD_BRCH_MAGIC;  // 将字段转换回主机字节序
        brch.fb_nchilds = nchilds;
        brch.fb_ctime = ctime;
        brch.fb_blkmap = blkmap;
        brch.fb_parent = parent;

        if (strcmp(brch.fb_name, bname) == 0) {  // 如果当前分支的名称与查找的分支名匹配
            sc->sc_branch = brch;  // 将找到的分支描述符存储到 `sc->sc_branch`
            sc->sc_branch_id = i;  // 保存当前分支的 ID
            return (0);  // 返回 0，表示成功找到分支
        }
    }

    return (ESRCH);  // 如果没有找到匹配的分支，返回 `ESRCH`（没有找到该项）
}
```

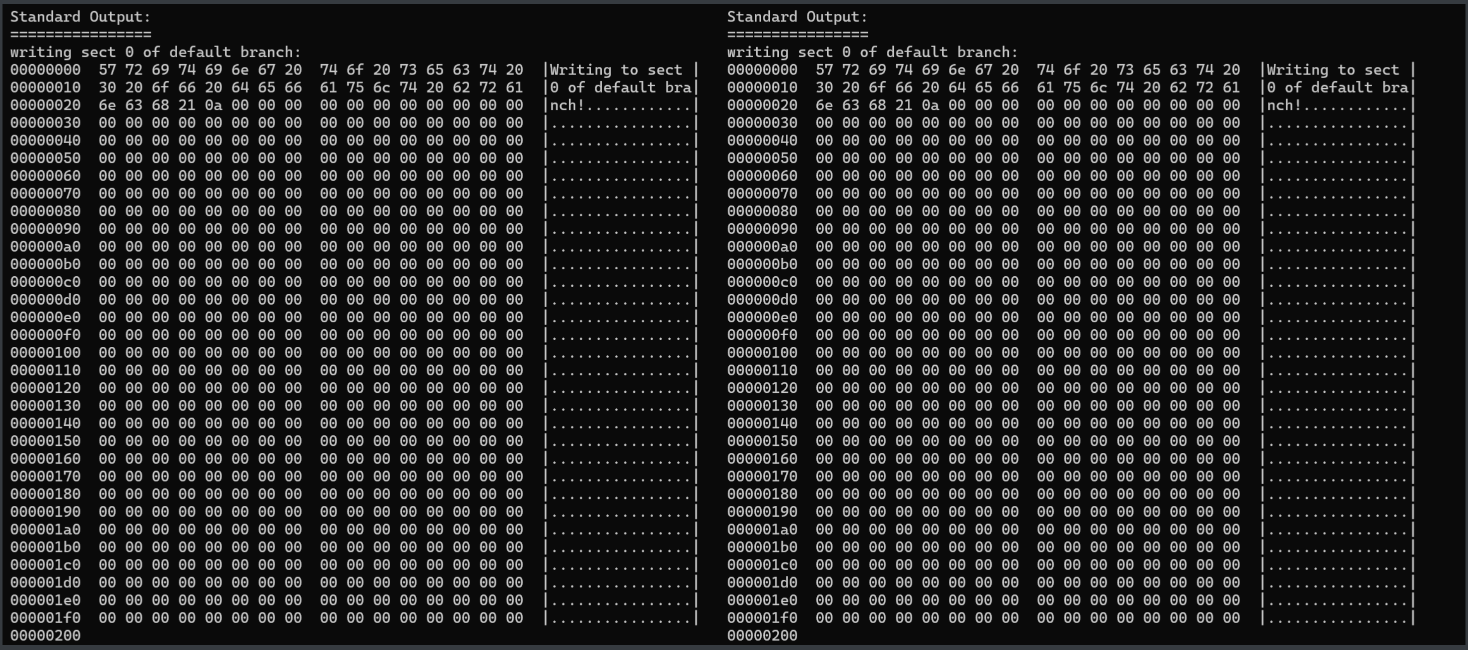
解决方法：

1. **检查分支创建的逻辑：**
   确保 `fvd_fork` 函数中的分支创建过程没有错误。你可以在该部分添加更多的错误检查和调试信息，查看是否在创建新分支时遇到错误。
2. **检查文件路径和挂载：**
   确认 `.fvd` 和 `.fvd.ref` 文件的位置是否正确，路径是否被正确配置并且挂载。
3. **调试缓存问题：**
   确保在每次写操作后，缓存被正确刷新（`fvd_cache_flush`），确保写入的数据正确地反映在磁盘镜像中。

如果你有更详细的错误日志或是具体的报错信息，我可以帮助你进一步诊断和解决问题。

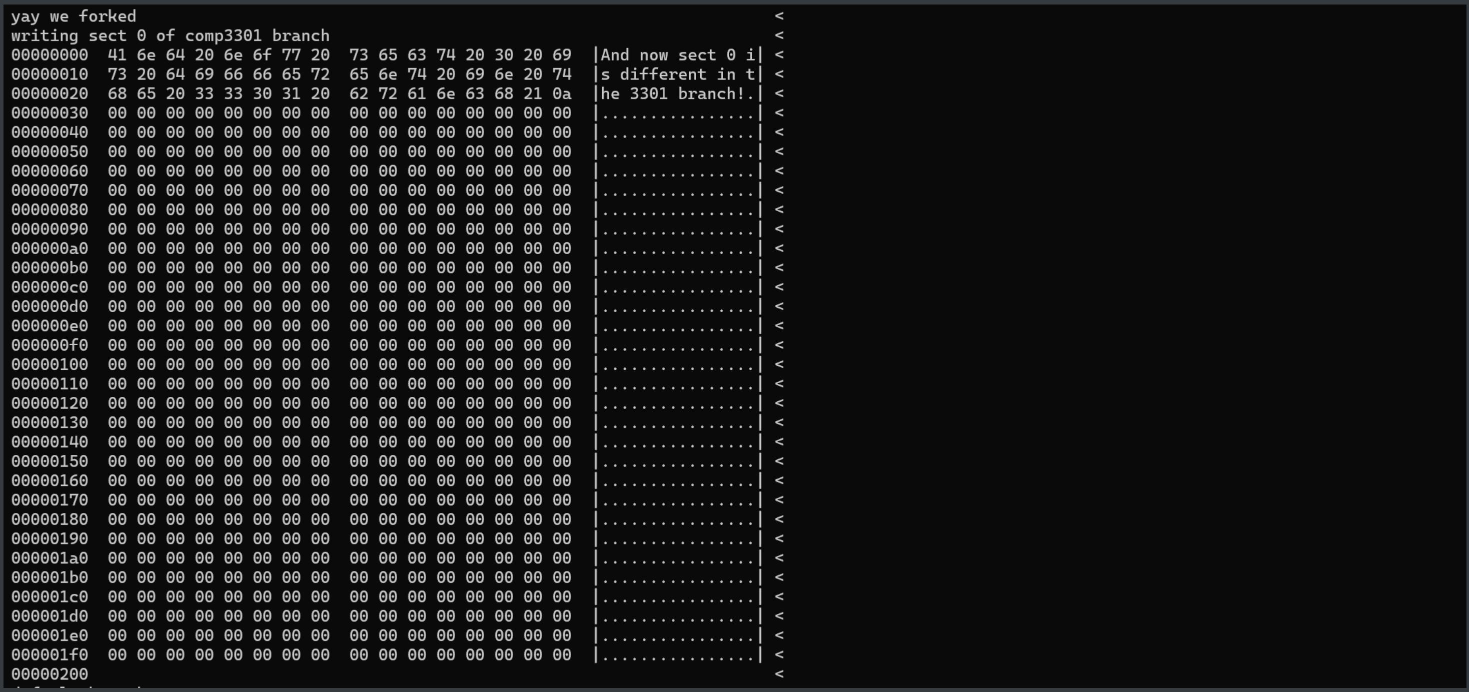


```
Test "3.2"...
---------------------------------- DESCRIPTION -----------------------------------
Writes sector 0 of the main branch, fork and write something else to the new
branch. Then check both branches and content should be different. This is done
on an image with sector 0 not allocated.
----------------------------------- LOCATIONS ------------------------------------
Test script:
    /usr/local/share/comp3301/a3/public/3.2.test

Expected output:
    /usr/local/share/comp3301/a3/public/3.2.outp
----------------------------------------------------------------------------------
```

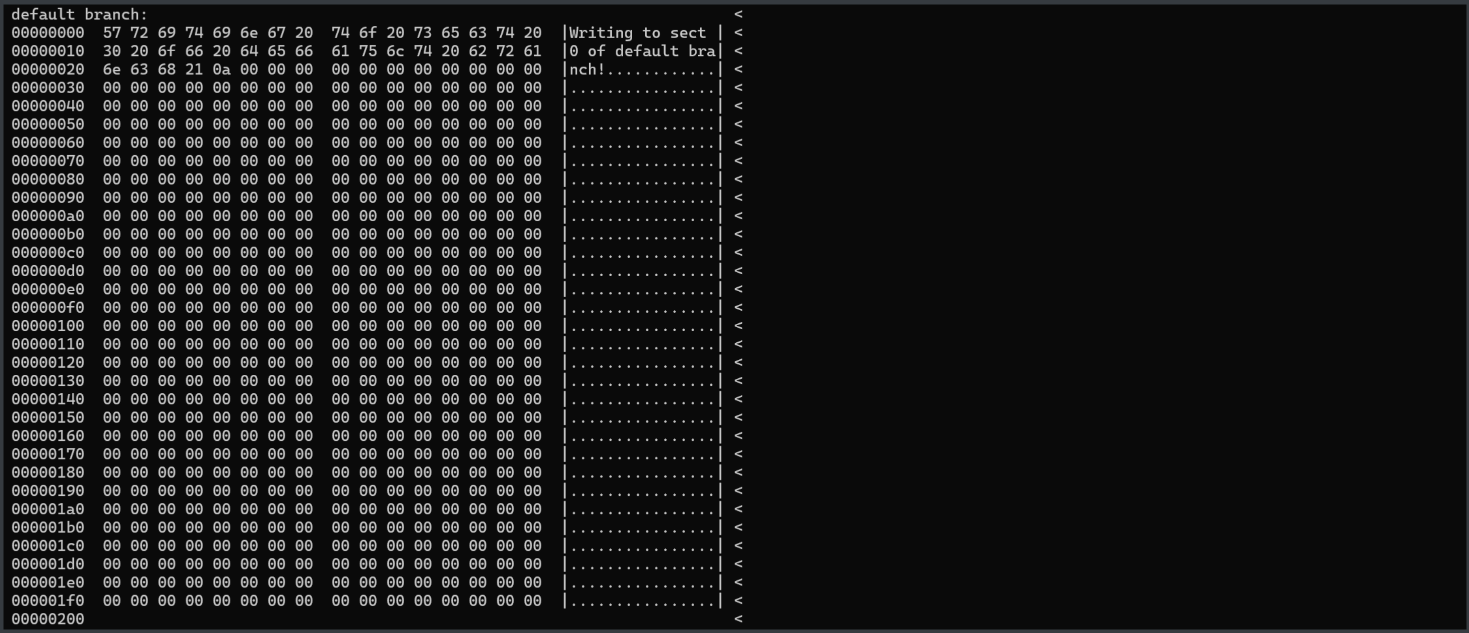


```
Standard Output:
================
writing sect 0 of default branch:
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect |
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra|
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200
```

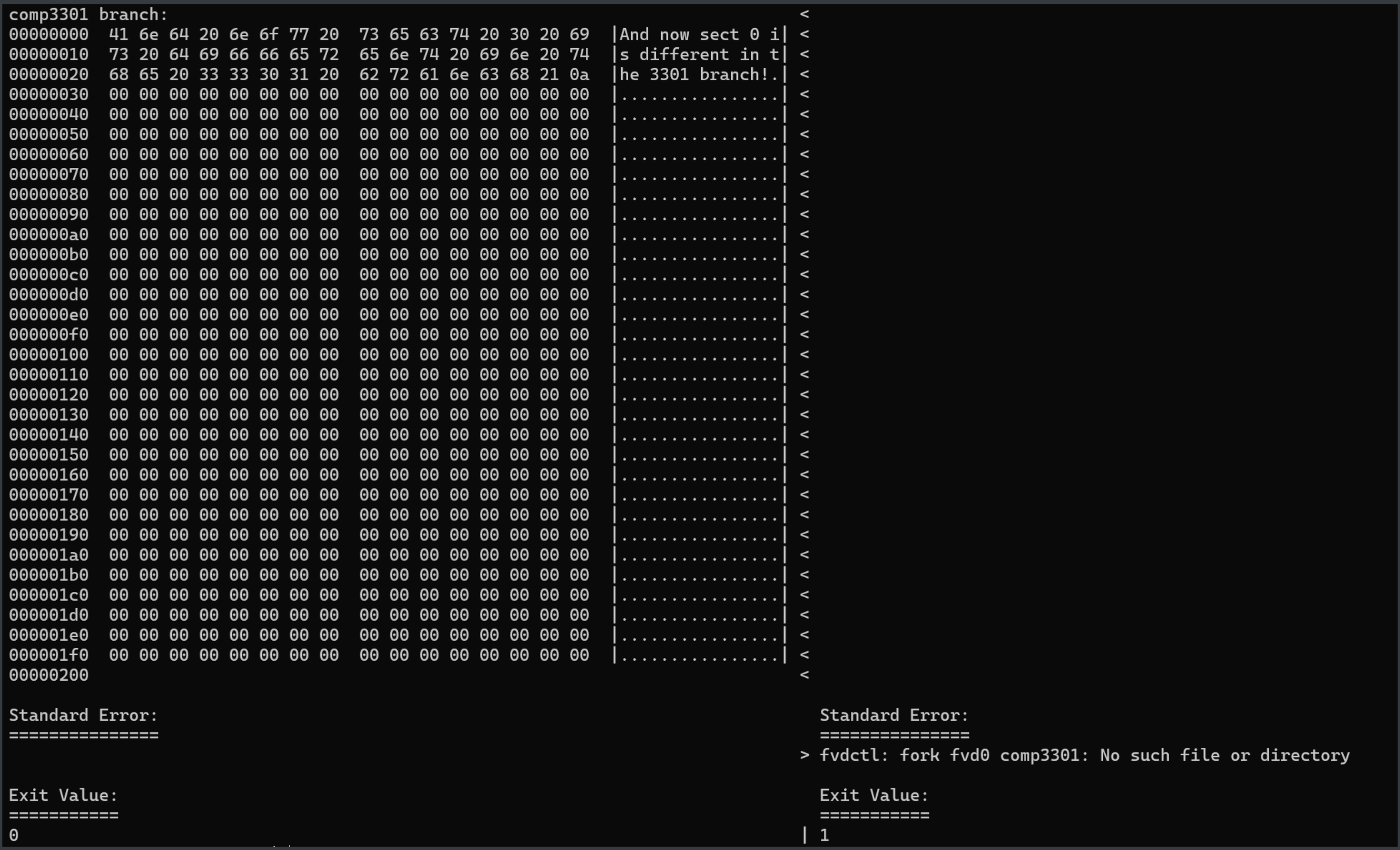
```
Standard Output:
================
writing sect 0 of default branch:
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect |
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra|
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200
```



```
yay we forked                                                                  <
writing sect 0 of comp3301 branch                                              <
00000000  41 6e 64 20 6e 6f 77 20  73 65 63 74 20 30 20 69  |And now sect 0 i| <
00000010  73 20 64 69 66 66 65 72  65 6e 74 20 69 6e 20 74  |s different in t| <
00000020  68 65 20 33 33 30 31 20  62 72 61 6e 63 68 21 0a  |he 3301 branch!.| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <
```



```
default branch:                                                                <
00000000  57 72 69 74 69 6e 67 20  74 6f 20 73 65 63 74 20  |Writing to sect | <
00000010  30 20 6f 66 20 64 65 66  61 75 6c 74 20 62 72 61  |0 of default bra| <
00000020  6e 63 68 21 0a 00 00 00  00 00 00 00 00 00 00 00  |nch!............| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <
```



```
comp3301 branch:                                                               <
00000000  41 6e 64 20 6e 6f 77 20  73 65 63 74 20 30 20 69  |And now sect 0 i| <
00000010  73 20 64 69 66 66 65 72  65 6e 74 20 69 6e 20 74  |s different in t| <
00000020  68 65 20 33 33 30 31 20  62 72 61 6e 63 68 21 0a  |he 3301 branch!.| <
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................| <
00000200                                                                       <

Standard Error:                                                                  Standard Error:
===============                                                                  ===============
                                                                               > fvdctl: fork fvd0 comp3301: No such file or directory

Exit Value:                                                                      Exit Value:
===========                                                                      ===========
0                                                                              | 1
```

根据你上传的截图和描述，`3.2` 测试似乎出现了一个问题，错误消息显示为：`fvdctl: fork fvd0 comp3301: No such file or directory`，并且退出值为 `1`。

从错误来看，问题的关键是找不到文件或目录。这通常意味着系统在尝试 `fork` 操作时无法找到指定的文件或路径。具体来说，错误的原因可能是：

1. **文件或路径缺失**：
   错误消息中提到 `comp3301` 文件没有找到，可能是该文件没有正确生成或路径配置错误。
2. **分支名称或映像路径问题**：
   可能在进行 `fork` 操作时，指定的分支 `comp3301` 不存在，或者文件路径不正确。

你可以从以下几个方面排查和解决问题：

1. 检查分支是否正确创建：

- 确保分支 `comp3301` 已经被创建，并且没有拼写错误。如果分支未正确创建，`fork` 操作就无法成功。

2. **确认路径和文件位置**：
   - 确认 `comp3301` 分支文件的路径是否正确配置，路径问题会导致文件找不到。检查路径中是否存在 `comp3301` 文件。

3. **查看 `fork` 操作的实现**：
   - 重新审查 `fvd_fork` 函数中如何处理分支文件的创建和映像路径，确保在 `fork` 时分支的路径和文件处理逻辑没有出错。

4. **检查磁盘映像是否正确加载**：
   - 确保磁盘映像被正确加载，路径和挂载是否正确。磁盘映像文件丢失或路径错误可能导致无法进行 `fork` 操作。

建议：

- **检查日志和错误信息**，确认缺失的文件或路径。
- **运行 `fvdctl` 和相关命令时，检查指定的路径是否正确**，例如检查分支文件 `comp3301` 是否存在。
- **验证分支创建和操作**，查看是否有任何错误或遗漏，导致 `fork` 操作失败。

如果你可以提供更多的错误日志或 `fvd_fork` 实现代码，我们可以进一步分析问题。