基于misc cgroup子系统实现对于cgroup级别打开文件数量限制

姓名：陈沛东

学号：121037910016

项目背景

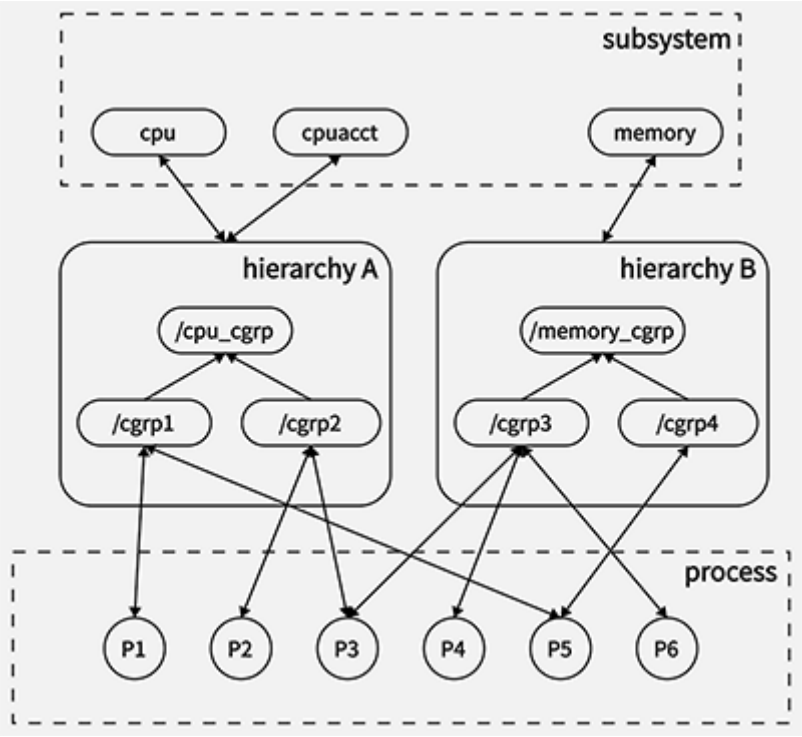
项目目的

在linux操作系统中，通过/proc/sys/fs/file-max可以获取全局最大打开文件数。但是针对cgroup级别，打开文件数目前是没有限制的。为了避免某些异常行为导致cgroup中打开文件数过多，从而中断了其他cgroup中的业务，所以需要对于cgroup级别进行打开文件数的上限的限制。misc cgroup子系统是5.13内核引入的一个限制和追踪某些资源的子系统，可以利用misc子系统对于cgroup级别进行打开文件数限制。

Cgroups

Cgroups是Linux内核提供的一种可以限制单个进程或多个进程所使用资源的机制，利用cgroups，我们可以在以进程（组）为单位分配系统资源，实现对诸如cpu、内存等系统资源的精细化控制。

在cgroups中，一个子系统（subsystem）对应着系统中一种可以控制的资源。例如cpu子系统可以用来控制了进程的cpu使用率，内存子系统可以用来控制进程的内存使用量。在cgroups的设计种，不同的cgroup以文件树的形式被组织成若干个森林，每个森林的根节点对应着一个层级结构（hierarchy），一个层级结构可以关联一个或几个子系统，来控制该层级结构cgroup中进程对相应资源的使用。每个进程可以位于不同层级结构的**某个**cgroup中（一个进程不能位于同一个层级结构的多个cgroup中）。进程、cgroup、层级结构和子系统的关系可以用下图来表示



Misc cgroup就是在linux 5.13后引入的一个与cpu、memory同级的子系统。在misc cgroup中，资源控制被抽象为两个数字：max（代表misc cgroup允许的最大资源使用量）；current（代表misc cgroup当前的资源使用量）。在运行过程中，所有位于misc cgroup中的进程必须保证current ≤ max。此外，对于父子关系的misc cgroup，子group中的资源使用量同时会累加到父group中。

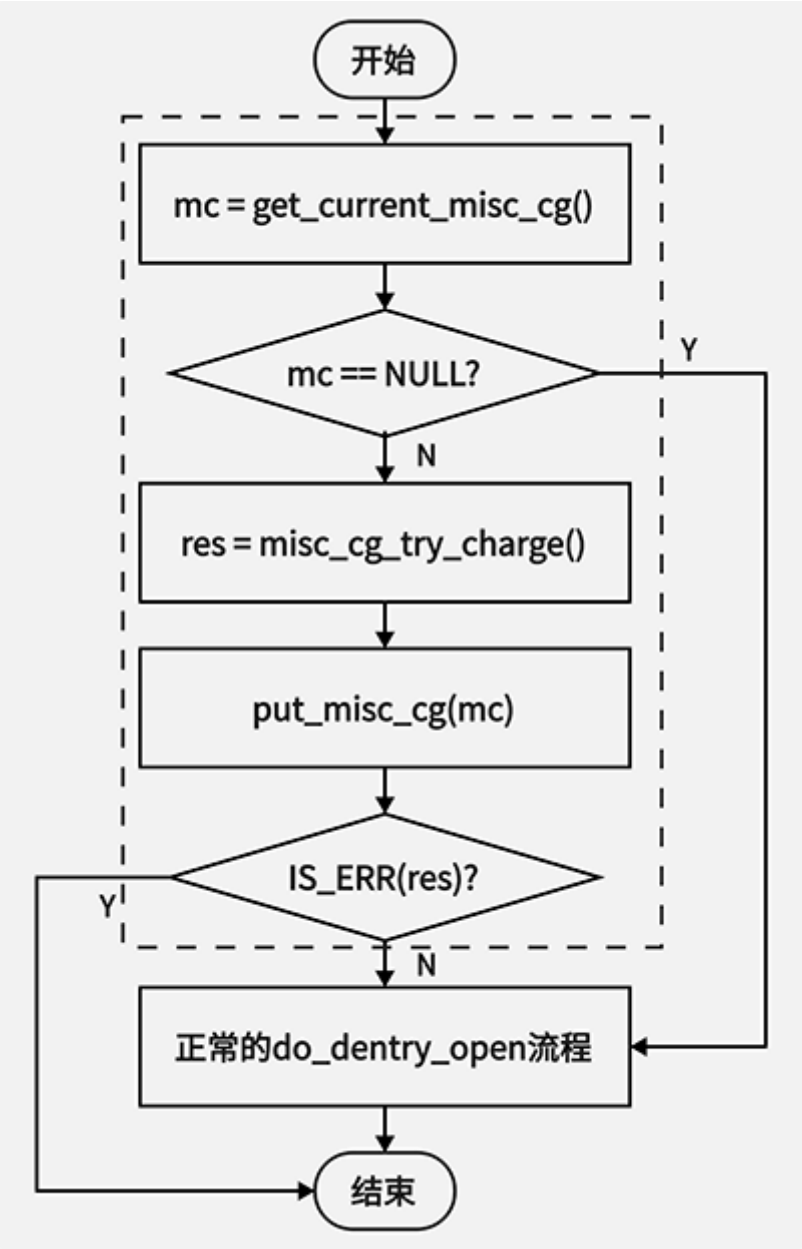
项目实现

MISC资源项添加

misc cgroup子系统通过名称来区分和管理不同的资源。本项目在/include/linux/misc_cgroup.h的enum misc_res_type中添加了一个枚举项：MISC_OPEN_FILE，表示用于管理文件打开数的misc cgroup资源；同时在/kernel/cgroup/misc.c的misc_res_name数组中添加了一个字符串元素："misc_open_file"，对应该资源的名称。

限制打开文件数

在linux中，程序打开文件的操作会经过一系列函数调用，最后调用到/fs/open.c中的do_dentry_open函数，在这个函数中调用具体文件系统的函数来打开文件。因此，本项目将misc cgroup限制进程打开文件数的代码逻辑实现在了do_dentry_open函数中，具体流程如下：

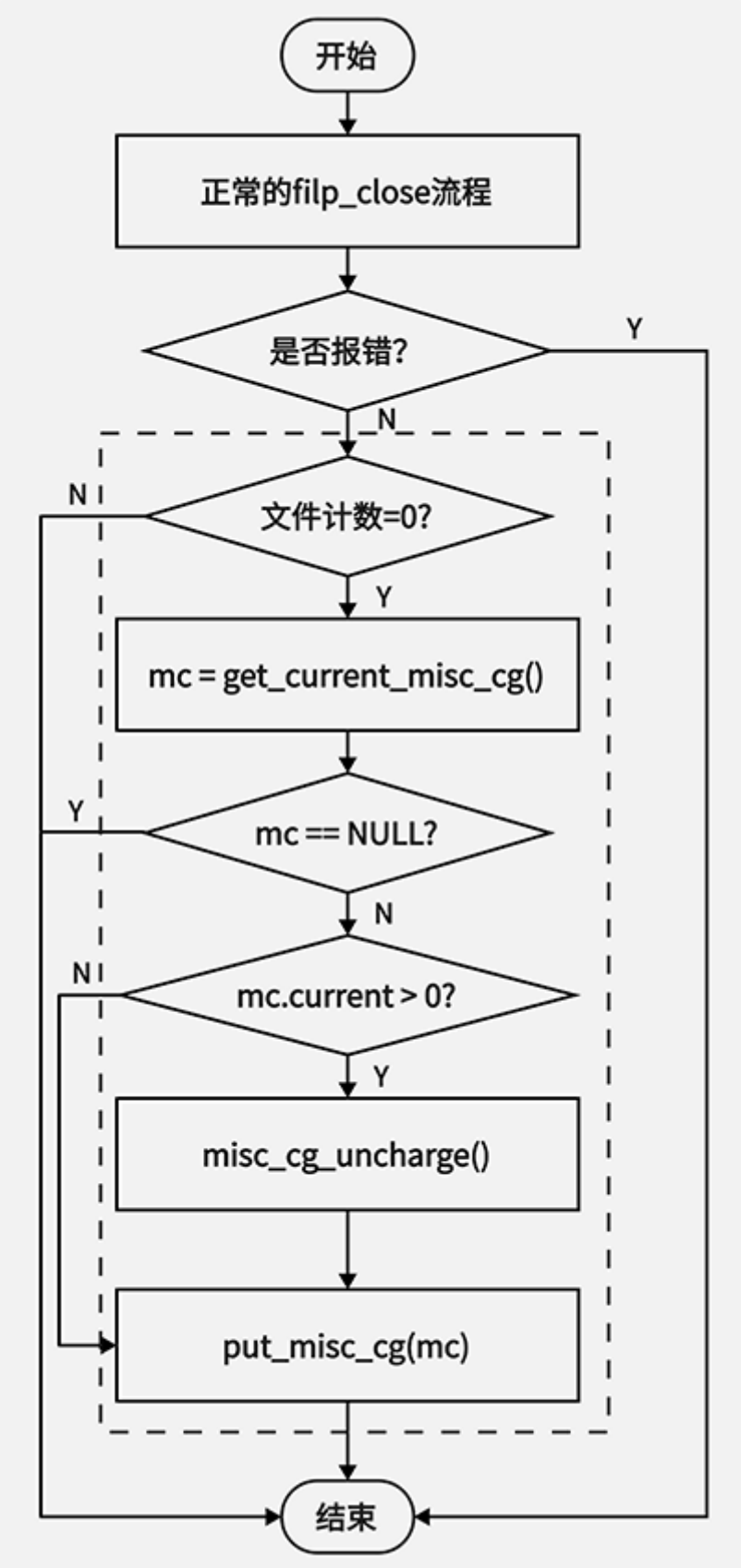


虚线框是本项目在do_dentry_open函数中添加的逻辑，用于限制进程打开的文件数不超过其所处misc cgroup的max值。

1. 调用get_current_misc_cg函数获取当前进程所在的misc cgroup，如果获取的结果为空，说明当前进程还没有加入misc cgroup的管控，跳转到步骤3；如果获取结果不为空，说明当前进程已经加入了misc cgroup的管控，跳转到步骤2；
2. 调用misc_cg_try_charge函数尝试对misc cgroup的current加1，代表将当前cgroup的资源使用量加1，然后调用put_misc_cg释放相应的数据结构资源。如果misc_cg_try_charge执行失败，说明当前进程所处的misc cgroup的打开文件数已经达到了该cgroup的最大值，无法再打开新的文件，返回错误；如果执行成功，跳转到步骤3；
3. 执行正常的dp_dentry_open流程，打开文件。

减少文件打开数

在open时需要增加文件打开数，那么相应的在close时就需要减少文件打开数。在linux中，程序关闭文件的操作会经过一系列函数调用，最后调用到/fs/open.c中的filp_close函数，在这个函数中执行关闭文件的操作。因此，本项目将减少misc cgroup打开文件数的代码逻辑实现在了filp_close函数中，具体流程如下：



虚线框是本项目在filp_close函数中添加的逻辑，用于在关闭文件时将misc cgroup的current值减1。

1. 多个进程可能同时打开了同一文件，所以我们首先判断文件的引用计数是否为0。如果不为0，则直接结束；如果等于0，则跳转到步骤2；
2. 调用get_current_misc_cg（get_current_misc_cg和put_misc_cg的作用与限制文件打开书中描述的一致，这里不再赘述），跳转到步骤3；

3. misc cgroup的current不能为负数，判断当前cgroup的current是否大于0。如果等于0，则直接结束；如果大于0，则跳转到步骤4；
4. 调用misc_cg_uncharge函数，将当前cgroup的current量减1。

设置capacity

在使用misc cgroup之前，需要设置单个cgroup所能承受的最大资源使用量。本项目在/init/main.c的start_kernel函数中，使用misc_cg_set_capacity对单个cgroup的最大子资源使用量进行了初始化，本项目使用的capacity为8192，即单个misc cgroup最多允许打开8192个文件。

错误信息反馈

在使用misc cgroup限制进程文件打开数的时候，可能会有异常情况产生，本项目针对这些异常给予了用户相应的反馈信息：

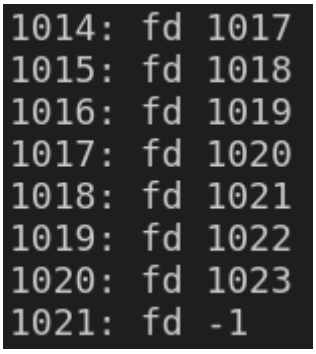
- 当进程打开的文件数超过了MAX(cgroup.max, capacity)时，misc_cg_try_charge函数会打印具体信息，包括导致打开文件数溢出的进程ID、cgroup的capacity以及文件数溢出的backtrace（有可能当前cgroup的祖先cgroup文件数溢出，backtrace会追踪到第一个文件数溢出的cgroup）
- 当用户使用echo命令/写文件向cgroup.max中写入值时，如果写入的值大于capacity，会提示参数不合法，max不能超出capacity，同时会给出capacity的值；如果写入的值小于cgroup.current，会提示参数不合法，max不能低于current，同时会给出current的值

项目测试

本项目通过在qemu中运行linux-5.18.6内核进行测试。本测试结果图中，所有的xxx: 都是 **0-base**。

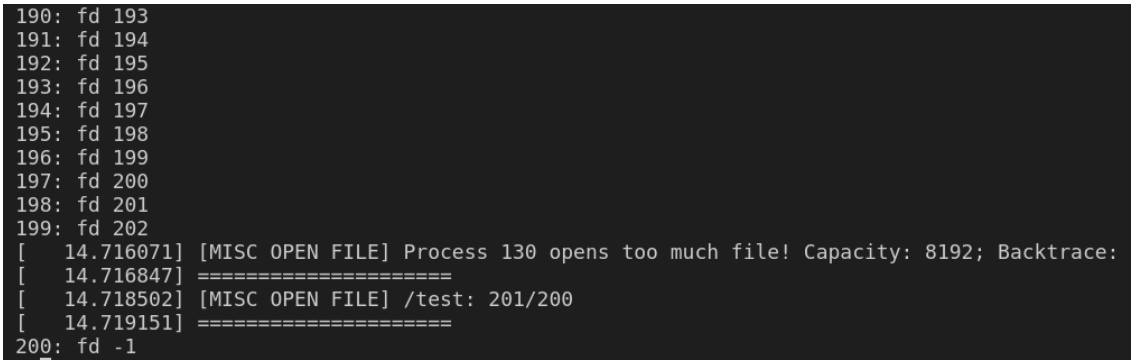
1. 不创建misc cgroup，运行程序打开2048个文件，期间不做关闭操作。

   结果：程序打开文件数到系统的软限制（在本项目中默认为1024）后，无法继续打开文件。linux系统中为每个进程的文件打开数规定了两个限制：软限制（soft limit）和硬限制（hard limit），前者是普通用户可以设置的对程序打开文件数的限制，后者是只有root用户可以设置的软限制的上限。一个程序所能打开的文件不能超过软限制；软限制可以通过 **ulimit -n xxx** 命令设置；软限制不能超过硬限制。（PS：图中的1021代表程序中使用了1021次open，之所以不能达到1024，是因为每个程序运行时默认都打开了标准输入、标准输出和标准错误，而这三者也被包含在软限制中）。



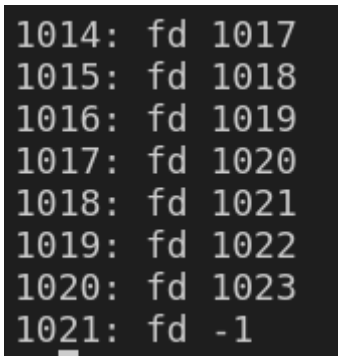
2. 设置单个misc cgroup，路径为/test，设置其max值为200，将程序pid写入/test/tasks，运行程序打开300个文件，期间不做关闭操作。

   结果：程序打开文件数到200（不包含标准输入、标准输出和标准错误，后续测试也是如此）后，无法继续打开文件。



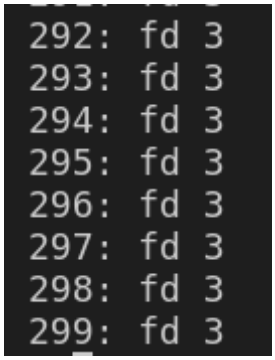
3. 设置单个misc cgroup，路径为/test，设置其max值为2000（大于软限制1024），将程序pid写入/test/tasks，运行程序打开300个文件，期间不做关闭操作。

   结果：程序打开文件数到软限制后，无法继续打开文件。



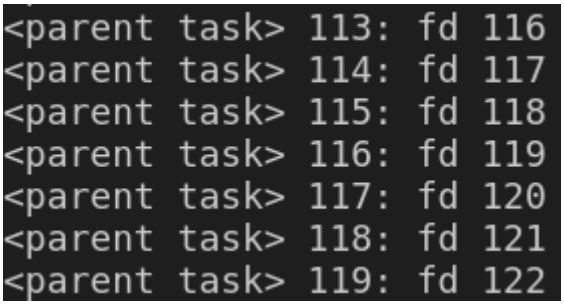
4. 设置单个misc cgroup，路径为/test，设置其max值为200，将程序pid写入/test/tasks，运行程序打开300个文件，期间每打开一个文件后，马上关闭。

   结果：程序可以正常打开300个文件。



5. 设置单个misc cgroup，路径为/test，设置其max值为200。将程序1的pid写入/test/tasks，运行程序1打开120个文件，期间不做关闭操作；然后在程序1中fork出程序2，运行程序2打开100个文件，期间不做关闭操作；程序1会等待程序2执行完毕后再退出。

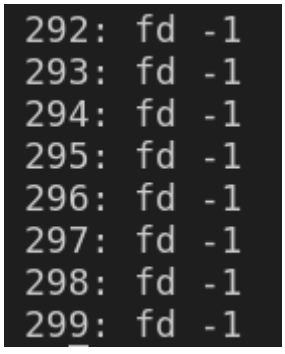
   结果：程序1可以打开120个文件，程序2只能打开80个文件。因为程序2是由程序1fork出来的，所以默认和程序1同属于/test的misc cgroup中，程序1已经打开了120个文件，所以程序2只能打开80个文件。




```
<child task> 74: fd 197
<child task> 75: fd 198
<child task> 76: fd 199
<child task> 77: fd 200
<child task> 78: fd 201
<child task> 79: fd 202
[   16.365418] [MISC OPEN FILE] Process 130 opens too much file! Capacity: 8192; Backtrace:
[   16.366993] ====================
[   16.367746] [MISC OPEN FILE] /test: 201/200
[   16.368201] ====================
<child task> 80: fd -1
```

6. 设置单个misc cgroup，路径为/test，设置其max值为200，将程序pid写入/test/tasks，运行程序打开300个不存在的文件，期间不做关闭操作。

   结果：程序可以正常运行300次open操作，因为打开不存在的文件不会增加misc cgroup的current。



7. 设置两个misc cgroup，路径为/test（g1）和/test/test-child（g2），设置g1.max = 200，g2.max = 300，将程序pid写入/test/test-child/taks，运行程序打开250个文件，期间不做关闭操作。

   结果：程序打开200个文件后，无法继续打开新的文件，因为g2中进程打开的文件数会累加到g1中。

```
193: fd 196
194: fd 197
195: fd 198
196: fd 199
197: fd 200
198: fd 201
199: fd 202
[  544.684430] [MISC OPEN FILE] Process 134 opens too much file! Capacity: 8192; Backtrace:
[  544.685134] ====================
[  544.685679] [MISC OPEN FILE] /test/test-child: 201/300
[  544.687088] [MISC OPEN FILE] /test: 201/200
[  544.687304] ====================
200: fd -1
```

8. 设置两个misc cgroup，路径为/test（g1）和/test/test-child（g2），设置g1.max = 500，g2.max = 300。将程序1的pid写入/test/test-child/taks，运行程序1打开250个文件，期间不做关闭操作；然后将程序2的pid写入/test/tasks，运行程序2打开400个文件，期间不做关闭操作；程序1会等待程序2执行完毕再退出。

   结果：程序1能够打开250个文件，但程序2只能打开250个文件而不是400个文件，理由同测试6。

```
<task in child cgroup> 243: fd 246
<task in child cgroup> 244: fd 247
<task in child cgroup> 245: fd 248
<task in child cgroup> 246: fd 249
<task in child cgroup> 247: fd 250
<task in child cgroup> 248: fd 251
<task in child cgroup> 249: fd 252
```

```
<task in parent cgroup> 244: fd 497
<task in parent cgroup> 245: fd 498
<task in parent cgroup> 246: fd 499
<task in parent cgroup> 247: fd 500
<task in parent cgroup> 248: fd 501
<task in parent cgroup> 249: fd 502
[   17.603859] [MISC OPEN FILE] Process 130 opens too much file! Capacity: 8192; Backtrace:
[   17.604692] ====================
[   17.605326] [MISC OPEN FILE] /test: 501/500
[   17.605952] ====================
<task in parent cgroup> 250: fd -1
```

9. 设置两个misc cgroup，路径为/test（g1）和/test1（g2），设置g1.max = 2000（超出软限制1024），g2.max = 1000。将程序1的pid写入/test/tasks，运行程序1打开2000个文件，期间不做关闭操作；将程序2的pid写入/test1/tasks，运行程序2打开1000个文件，期间不做关闭操作；程序1和程序2并行执行。

   结果：程序1打开文件数到软限制后，无法继续打开文件；程序2可以打开1000个文件。两个程序位于同级的cgroup中，互不影响。

```
<task in test cgroup> 1014: fd 1017
<task in test cgroup> 1015: fd 1018
<task in test cgroup> 1016: fd 1019
<task in test cgroup> 1017: fd 1020
<task in test cgroup> 1018: fd 1021
<task in test cgroup> 1019: fd 1022
<task in test cgroup> 1020: fd 1023
<task in test cgroup> 1021: fd -1
```

```
<task in test1 cgroup> 994: fd 997
<task in test1 cgroup> 995: fd 998
<task in test1 cgroup> 996: fd 999
<task in test1 cgroup> 997: fd 1000
<task in test1 cgroup> 998: fd 1001
<task in test1 cgroup> 999: fd 1002
```

10. 设置一个misc cgroup，路径为/test，向其中的max写入10000（大于capacity8192）。

    结果：写入失败，无法向一个misc cgroup的max中写入比capacity还要大的值。

```
[  317.232801] [MISC OPEN FILE] Misc.max 10000 is too big for /test, misc.capacity: 8192
```

11. 设置一个misc cgroup，路径为/test，设置其max值为500，将程序pid写入/test/tasks，运行程序打开300个文件，期间不做关闭操作，然后向/test的max写入200。

结果：写入失败，无法向一个misc cgroup的max写入比current还要小的值。

```
291: fd 294
292: fd 295
293: fd 296
294: fd 297
295: fd 298
296: fd 299
297: fd 300
298: fd 301
299: fd 302
[  450.980408] [MISC OPEN FILE] Misc.max 200 is too small for /test, misc.current: 301
```