OSlab2 实验报告

161220117 唐诗美

一、测试用法

这个实验一共设计对文件单线程打开的测试和多个线程同时访问一个文件测试。使用方法如下：

在./src/test.c 里面的最下方的函数 void test_file()中

```
void test_file()
{
    kmt->spin_init(&lk, "filetest_lk");
    kmt->spin_init(&lk_thread, "multithread_lk");
    single_thread_test();
    //multi_thread_test();
    return;
}
```

有 single_thread_test()和 multi_thread_test()两个测试函数。测试 single_thread_test()的时候需要把 multi_thread_test()注释了，测试 multi_thread_test()需要把 single_thread_test()注释了。

single_thread_test()的内容以及输出内容的解释如下：

它创建了三个线程分别测试/dev /proc /这三个虚拟文件系统的基本使用情况。

对于/dev，进行/dev/random 测试，可从输出中看见产生了（伪）随机数；进行/dev/null 测试，分别从向其中写东西以及读取，从输入中可看出什么都没有；对于/dev/zero 的测试与/dev/null 基本同理；

对于/proc，进行/proc/cpuinfo 测试，可看见输出 cpuinfo 里面（自己填也不知道那些参数应该是什么）的内容，进行/proc/meminfo 的测试，与 cpuinfo 同理，输出可见；进行/proc/0 测试，也就是创建的第一个线程的信息开始读取，（是从 0 开始进行线程计数的…），可看见线程相关信息的输出；

对于/，进行创建文件写文件读文件的操作。首先创建/forty/40c 文件，向其中写入“forty-forty\nthis is a test for kvdb\n40404040\n”的内容，然后会输出。

以上步骤是测试基本的虚拟文件系统的实现，以及 open()、read()、write()、lseek()、close()的基本实现问题。

测试的输出解释如下：

(1)procfs

```
in kmt thread_cnt:1
file_open:inode->name:/proc/0
in file_close:fd:3
in kmt thread_cnt:2
file_open:inode->name:/proc/1
in file_close:fd:3
in kmt thread_cnt:3
file_open:inode->name:/proc/2
in file_close:fd:3
```

这部分是线程的创建。

```
old id:1 current id:2
in os_interrupt eip:0x0010629a
request trap into kernal...
```

以后出现的这一部分都是线程切换的过程。为了方便调试输出。

```
proc test: the cpu fd is 3
proc_test:size:28
content:
My cpuinfo:remain to be done
in file_close:fd:3
proc_test: the mem fd is 3
proc_test:size:28
content:
My meminfo:remain to be done
in file_close:fd:3
proc_test: the process fd is 3
proc_test:size:65
content:
id: 0
stack size: 262144
stack start: 2143862 stack end: 2406006
in file_close:fd:3
proc_test:end
```

打开 cpuinfo 里面的信息是”My cpuinfo:remain to be done”同时输出打开和关闭时的 fd。打开 meminfo 里面的信息是”My meminfo:remain to be done”同时输出打开和关闭时的 fd。打开/proc/0 的进程的信息，id 是这个进程对应的 pid，同时又栈的大小和起始结束地址等。最后结束 proc_test 函数时输出”proc_test:end”

(2)kvfs

```
the file has not been created!!
kv_test:create!!!
file_open:inode->name:/forty/40c
kv_test:fd for /forty/40c:3
kv_test:write /forty/40c size:45
kv_test:read /forty/40c size:45
content:
forty-forty
this is a test for kvdb
40404040
in file_close:fd:3
kv_test:end
```

打开文件判断是否有创建，如果没有则创建一份 `kv_test:create!!!` 就是这一句，如果已经创建存在了就是 `kv_test:not create!!` 这一句。然后在 open 函数里面输出 inode 的名字；输出打开文件的 fd，向其中写入一串字符串，再去读取它，输出读取的内容，最后关闭。

(3)devfs

```
dev_test:random fd:3
dev_test:this is the random number return by /dev/random:621 size:0
dev_test:this is the random number return by /dev/random:4524 size:0
dev_test:null fd:4
in file_write:name:/dev/null buf:40404040 size:8
dev_test:this is the writing /dev/null operation return value:8
dev_test:after read /dev/null buf:0
in file_close:fd:4
dev_test:zero fd:4
dev_test:this is buf in zero before read buf:40404040
dev_test:after read /dev/zero buf:0
in file_close:fd:4
in file_close:fd:3
dev_test:end
```

打开文件。打开第一个/dev/random 文件后先不关闭，看后面的 fd 是否正常工作。可以看出输出的随机数。接着再对 null 和 zero 的测试，可以看出都没有输出。如果一开始 buf 中有内容为”40404040”，读取 zero 后就变为什么都没有了。最后关闭文件。

multi_thread_test()测试内容如下：

这个函数中创建了四个进程，其中每两个进程是对同一个文件进行读写操作。在这里称为 file1, file11（与 file1 访问同一文件）, file2, file22(（与 file2 访问同一文件）。对它们进行一些读写操作最后看输出结果。

测试的输出解释如下：

(1)file1

这里的 file1 是打开一个/home/forty 的文件，并在文件中写入”this is /home/forty”的内容。

(2)file11

这里是 file11 打开的内容，与 file1 同理。

可以看出它们之间进行了线程的切换。

(3)file2

file2 是打开/home/4040 的一个文件，首先是向其中写入”this is /home/4040”的文字，然后再第二次写入相同的话。读取的时候先把 offset 设为从头开始，读取 19 个字节的内容。第二次同样的，但是是读取 38 个字节的内容。最后关闭文件

(4)file22

file22 是打开/home/4040 的一个文件，首先是向其中写入”this is /home/4040”的文字，然后再第二次写入相同的话。读取的时候先把 offset 设为从头开始，读取 19 个字节的内容。第二次同样的，但是是读取 38 个字节的内容。最后关闭文件。

（万一…万一测试不对……mul_log 和 single_log 有记录这些信息的）

二、实验心得

这次实验一开始拿到实验讲义很懵，不知道应该干什么，查了很多资料，甚至以为还有实现 dentry_t 这种类型，觉得很不明觉厉。也不知道这个文件系统到底要实现成什么样。特别是 cpuinfo 和 meminfo 那些，因为这些硬件的内容这个 OS 似乎并不支持？所以就很懵了。后来通过询问同学，大概了解了一点思路。

实现的过程其实不是很难，就是复制粘贴很多相同的内容。但是 debug 真是非常痛苦了。有由于手误在判断的时候少加了一个感叹号的，有发现以前写的 strcmp 和 strcat 写得不对的。不过最最最最坑的地方是我发现我 pmm->free 实现得不对，brk 的实现也有问题。（这个 bug 的发现过程真的是很纠结了…起初是发现会段错误，有时候本来是可以打开的文件结果就变成不存在了。就很懵，然后各种 printf 终于发现在执行了一个 strcpy 之后另外一个变量的值也被修改了，还有一个地方是发现分配了内存之后没有回收…）终于定位到是 free 写错了，修改了之后我也不知道那个时候我哪里的实现还是有问题，就是不对。然后我就打算用数组来写这个实验，不用 pmm 了，改了半天，过程中出现一堆编译错，接着最后测试的时候发现堆的 start 和 end 居然到了同一个地方。怀疑可能时.data 段太大了。然后又还是改回了分配指针的。通过整理思路好好修改了一遍终于，我觉得应该还是能经得住一点测试了。