Lab5: 文件系统与SHELL

author：魏新鹏

student ID：519021910888

1 练习题1

实现位于 userland/servers/tmpfs/tmpfs.c 的 tfs_mknod 和 tfs_namex。

- tfs_mknod: 根据mkdir判断新建普通文件还是目录的inode，然后新建dent加入该文件所在目录的hash_list。
- tfs_namex:遍历文件名，以`/`为界，找到逐级的目录，然后依次调用tfs_lookup，若碰到没有找到的情况，根据是否mkdir_p来判断是否要补全空缺的目录项，最后一个文件项（通过`**name == '\0'`来判断）不用调用tfs_lookup，直接修改leaf即可。

2 练习题2

实现位于 userland/servers/tmpfs/tmpfs.c 的 tfs_file_read 和tfs_file_write 。提示：由于数据块的大小为PAGE_SIZE，因此读写可能会牵涉到多个页面。读取不能超过文件大小，而写入可能会增加文件大小（也可能需要创建新的数据块）。

- tfs_file_read: 根据offset得到page number和page offset，然后找到radix tree中的数据页，通过memcpy拷贝至buffer中。因为最多只能拷贝PAGE_SIZE大小，所以要用一个while循环检测size是否大于0。
- tfs_file_write: 与上类似，只不过当page为空时要malloc一个新页。

3 练习题3

实现位于 userland/servers/tmpfs/tmpfs.c 的 tfs_load_image 函数。需要通过之前实现的tmpfs函数进行目录和文件的创建，以及数据的读写。

- tfs_load_image: 遍历cpio文件，先调用tfs_namex找到它的parent dir和文件名，然后调用tfs_lookup判断改文件是否存在，如果不存在，则根据File type调用tfs_creat或者tfs_mkdir新建。最后将数据写入。

4 练习题4

利用 userland/servers/tmpfs/tmpfs.c 中已经实现的函数，完成在 userland/servers/tmpfs/tmpfs_ops.c 中的 fs_creat 、tmpfs_unlink 和tmpfs_mkdir 函数，从而使 tmpfs_* 函数可以被 fs_server_dispatch 调用以提供系统服务。对应关系可以参照 userland/servers/tmpfs/tmpfs_ops.c 中 server_ops 的设置以及userland/fs_base/fs_wrapper.c 的 fs_server_dispatch 函数。

- fs_creat: 首先调用tfs_namex找到parent dir和文件名，然后调用tfs_creat新建即可。
- tmpfs_unlink: 首先调用tfs_namex找到parent dir和文件名，然后调用tfs_remove删除即可。
- tmpfs_mkdir: 首先调用tfs_namex找到parent dir和文件名，然后调用tfs_mkdir新建文件夹即可。

5 练习题5

实现在 userland/servers/shell/main.c 中定义的 getch ， 该函数会每次从标准输入中获取字符，并实现在 userland/servers/shell/shell.c 中的 readline ， 该函数会将按下回车键之前的输入内容存入内存缓冲区。代码中可以使用在 libchcore/include/libc/stdio.h 中的定义的I/O函数。

- getch: 直接调用getc从串口读入一个字符。
- readline: 反复调用getch读入字符，如果是`\t`则调用do_complement进行补全，如果是`\r`或者`\n`则不加入buffer，默认行为是加入buf并将打印至标准输出。

6 练习题6

根据在 userland/servers/shell/shell.c 中实现好的 bultin_cmd 函数，完成shell中内置命令对应的 do_* 函数，需要支持的命令包括： ls [dir] 、 echo [string] 、 cat [filename] 和 top。

- do_ls: 略过开头的ls两个字母与空格然后调用fs_scan。fs_scan中首先open对应的dir拿到fd，然后调用demo_getdents打印出目录中的所有文件。
- do_echo: 略过开头的echo四个字母与空格然后将cmdline打印即可。
- do_cat: 略过开头的cat三个字母与空格然后调用print_file_content打印文件内容，其中print_file_content首先open file拿到获得fd，然后反复读取文件直到ipc返回的文件长度为0。
- do_top: 直接调用chcore_sys_top即可。

7 练习题7

实现在 userland/servers/shell/shell.c 中定义的 run_cmd ， 以通过输入文件名来运行可执行文件，同时补全 do_complement 函数并修改 readline 函数，以支持按tab键自动补全根目录（/）下的文件名。

- do_complement: 首先open根目录拿到fd，然后根据complement_time和offset反复调用get_dent_name获取目录项的name，将其打印即可。（提醒：要略过`.`项）

8 练习题 8

补全 userland/apps/lab5 目录下的 lab5_stdio.h 与 lab5_stdio.c 文件，以实现fopen , fwrite , fread , fclose , fscanf , fprintf 五个函数，函数用法应与libc中一致，可以参照 lab5_main.c 中测试代码。

- fopen: 先调用open，若ret>0则打开成功，若ret == -2，则根据filename新建文件，然后再调用open。
- fwrite: 直接构造FS_REQ_WRITE类型的ipc_call即可。
- fread: 直接构造FS_REQ_READ类型的ipc_call即可。
- fclose: 直接构造FS_REQ_CLOSE类型的ipc_call即可，记得free FILE对象。
- fprintf: 调用simple_vsprintf获取要写入文件的内容，然后调用fwrite将其写入即可。
- fscanf: 先调用fread读入文件的所有内容到buf，然后同时遍历fmt和buf，当fmt碰到`%`根据下一个字符是s还是d进行处理。两种情况都首先用va_arg获取指针，如果是s，则一直读到空格，如果是d，则一直读到不属于`'0' ~ '9'`。将对应的指针赋成相应的变量即可。

9 练习题9

FSM需要两种不同的文件系统才能体现其特点，本实验提供了一个fakefs用于模拟部分文件系统的接口，测试代码会默认将tmpfs挂载到路径 / ，并将fakefs挂载在到路径 /fakefs 。本练习需要实现 userland/server/fsm/main.c 中空缺的部分，使得用户程序将文件系统请求发送给FSM后， FSM根据访问路径向对应文件系统发起请求，并将结果返回给用户程序。实现过程中可以使用 userland/server/fsm 目录下已经实现的函数。

CREAT, OPEN的处理策略类似：

1. 调用get_mount_point得到mpinfo，这其中包含了ipc_struct，也就是ipc的client。
2. 调用strip_path得到path在fs root中的绝对路径。
3. 根据请求类型构造对应的ipc_call发往mpinfo中对应的fs server。

CLOSE, GETDENTS64, WRITE以及READ的处理策略类似：

1. 根据fd和client_badge得到相应的mpinfo，其中包含了ipc_struct，也就是ipc的client。
2. 根据请求类型构造对应的ipc_call发往mpinfo中对应的fs server。

10 练习题 10

为减少文件操作过程中的IPC次数，可以对FSM的转发机制进行简化。本练习需要完成 libchcore/src/fs/fsm.c 中空缺的部分，使得 fsm_read_file 和 fsm_write_file 函数先利用ID为FS_REQ_GET_FS_CAP的请求通过FSM处理文件路径并获取对应文件系统的 Capability，然后直接对相应文件系统发送文件操作请求。

- fsm_read_file: 首先根据path调用get_fs_cap获取对应fs server的cap。其中get_fs_cap就是往fsm_server发FS_REQ_GET_FS_CAP类型的ipc_call。然后调用get_fs_cap_info建立fs_cap到fs_cap_info_node的对应关系。然后先调用open_file获取fd，然后构造read file的ipc请求，最后close_file。注意：最后的这三部操作都是通过特定的fs_server的client发送的，而不是fsm_server。
- fsm_write_file: 与上类似。