Lab 5 Exam

准备工作：创建并切换到 `lab5-exam` 分支

请基于已完成的 lab5 提交自动初始化 lab5-exam 分支， 然后在开发机依次执行以下命令：

```
$ cd ~/ 学号
$ git fetch
$ git checkout lab5-exam
```

初始化的 `lab5-exam` 分支基于课下完成的 `lab5` 分支，并且在 `tests` 目录下添加了 `lab5_find` 样例测试目录。

题目背景 & 题目描述

在 Linux 系统中， `find` 命令是一条很常用的命令，它支持根据多种条件筛选文件，并把文件的路径输出。比如根据路径、名字来匹配相应的文件：

```
find / -name "motd"
```

该命令在根目录下查找名为 `motd` 的文件，并把找到的所有文件的路径输出。

在本次测试中，我们希望同学们实现一个支持以基础路径和文件名为条件查找文件的简易版函数
`int find(const char *path, const char *name, struct Find_res *res)` 。该函数在 `path` （都以 **绝对路径** 给出）对应的文件及其子目录中递归查找名字为 `name` 的文件，并把查到的文件的 **绝对路径** 和文件总数结果存入 `res` 指向的预定义结构体，只有文件名和 `name` 完全一致才视作匹配，无需支持通配符。

`struct Find_res` 定义在了 `user/include/lib` 中， `file_path` 用于存查到的路径， `count` 用于存查到的文件数量。我们保证在本次测试中，该结构体足够装下所满足条件的路径，不会溢出。同时你需要注意我们修改了 `MAXPATHLEN` ，本题中你需要用宏来表示最大路径长度。

```
#define MAXPATHNUM ((PAGE_SIZE - sizeof(int)) / MAXPATHLEN)
struct Find_res {
```

```
    char file_path[MAXPATHNUM][MAXPATHLEN];
    int count;
};
```

函数正常执行则返回 0，错误时对应的负错误码，本题保证只会出现两种错误，均已经在 `lab5` 课下实现，即查找 `path` 对应文件的时候没找到或者路径太长。**如果在 `path` 对应的文件目录下查找名字为 `name` 的文件过程中发现路径过长，则跳过该路径继续处理其他路径即可，不视作错误**

需要注意的是，本函数的匹配的规则和标准 `find <path> -name "<name>"` 实现一致，你可以在跳板机上尝试运行 `find` 命令，查看一些特殊情况下标准行为（例如 `<path>` **定位的文件名字恰为** `<name>`**，提示：在本题中，这种情况下你需要把这个文件包括在查询结果中）**，**为了简化实现，我们保证测试点中** `<path>` **对应的文件如果能够找到，一定是目录类型，同时，为了简化实现，我们保证所有测试点中，** `path` **均以路径分隔符** `/` **结尾**。

实现思路

- 在 `user/include/lib.h` 中我们已经给出了所需的查询结果结构体，你无需再复制

```
#define MAXPATHNUM ((PAGE_SIZE - sizeof(int)) / MAXPATHLEN)
struct Find_res {
    char file_path[MAXPATHNUM][MAXPATHLEN];
    int count;
};
```

你可以使用文件系统服务进程来完成整个查找过程。可以在用户函数中添加如下请求接口:

- 在 `user/include/lib.h` 的 `file.c` 注释下新增声明

```
int find(const char *path, const char *name, struct Find_res *res)
```

- 将该函数 `find` 的实现复制到 `user/lib/file.c`

```
int find(const char *path, const char *name, struct Find_res *res) {
    return fsipc_find(path, name, res);
}
```

(**请注意，本次** `lab5-exam` **只需要通过编译，就可以通过第一个测试点并拿到 50 分，如果后续完成确实有困难，你需要根据上面的思路添加** `find` **函数的声明和实现，并把** `find` **的函数体内部** `return fsipc_find(path, name, res);` **改成**

`return 0;` **，这样能用最少的行数通过编译，并通过第一个测试点。如果想完整实现本题目要求的所有功能，请忽略这一段话**)

- 在 `user/include/fsreq.h` 中的枚举类型中增加一个对于文件系统的请求类型 `FSREQ_FIND` **，请注意要把它放在** `MAX_FSREQNO` **前**
- 在同一文件 `user/include/fsreq.h` 里面加上请求结构体

```
struct Fsreq_find {
        char req_path[MAXPATHLEN];
        char req_name[MAXNAMELEN];
};
```

- 在 `user/include/lib.h` 的 `fsipc.c` 的注释下新增声明

```
int fsipc_find(const char *path, const char *name, struct Find_res *res);
```

- 在 `user/lib/fsipc.c` 中将该函数的实现复制过去

```
int fsipc_find(const char *path, const char *name, struct Find_res *res) {
    if (strlen(path) == 0 || strlen(path) > MAXPATHLEN) {
            return -E_BAD_PATH;
    }
    struct Fsreq_find *req = (struct Fsreq_find *)fsipcbuf;
    strcpy((char *)req->req_path, path);
    strcpy((char *)req->req_name, name);

    int r = fsipc(FSREQ_FIND, req, res, 0);
    return r;
}
```

然后在文件系统服务函数中实现服务接收和递归查找：

- 仿照其他项，在 `fs/serv.c` 的服务函数分发表 `serve_table` 最后新增一项

```
    [FSREQ_FIND] = serve_find,
```

- 在 `fs/serve.c` 的分发表上方将 `serve_find` 函数的实现复制过去

```
void serve_find(u_int envid, struct Fsreq_find *rq) {
    struct Find_res res __attribute((aligned(PAGE_SIZE))) = {};
    int r = find_files(rq->req_path, rq->req_name, &res);
    if (r) {
        ipc_send(envid, r, 0, 0);
    }
```

```
    else {
        ipc_send(envid, 0, (const void*)&res, PTE_D | PTE_LIBRARY);
    }
}
```

- 在 fs/serv.h 中添加声明

```
int find_files(const char *path, const char *name, struct Find_res *res)
```

- 然后在 fs / fs.c 中完成 fs / find_files 函数，实现本服务的核心功能。你可以先使用 walk_path 函数找到 path 对应的文件夹，然后再文件夹下寻找名字恰为 name 的文件。**你可以参照 fs /fs.c 文件里的 dir_lookup 学会对文件的各个块进行访问，不过需要注意 dir_lookup 不支持递归查找，你需要用深度优先搜索支持递归查找**(下一条提示给出了参考实现)

```
int find_files(const char *path, const char *name, struct Find_res *res) {
        struct File *file;
        // 用 walk_path 来找到 path 对应的文件夹
        // Lab5-Exam: Your code here. (1/2)

        // 在 path 对应的文件夹下面遍历，找到所有名字为 name 的文件，你可以调用下面的参考函
        // Lab5-Exam: Your code here. (2/2)
}
```

- 你可以用任意正确的方式实现 (2/2) 部分，这里用一个提供了一个参考实现函数，你只需要按注释填写四个空

```
int traverse_file(const char *path, struct File *file, const char *name, struct Find_

    u_int nblock;
    nblock = file->f_size / BLOCK_SIZE;

    // 1. 检查路径长度是否符合要求，如不符合，直接返回
    if (/* path 的长度为零或不小于最大路径长度*/) {
        /*返回*/
    }

    // 2. 比较当前文件名是否等于 name，如果相等则更改 res
    if (/*file 的名字等于 name*/) {
        /*增加 res->count*/
        /*添加 res 的路径*/
    }
    if (file->f_type == FTYPE_DIR) {
        for (int i = 0; i < nblock; i++) {
            void *blk;
            try(file_get_block(file, i ,&blk));
```

```
            struct File *files = (struct File *)blk;

            for (struct File *f = files; f < files + FILE2BLK; ++f) {
                char curpath[MAXPATHLEN + MAXNAMELEN + 1];
                // 3. 把 path 和 name 拼接起来得到下一层文件路径，注意结尾的 '\0'
                // 提示：我们没有实现 strcat 工具函数，你可以用 strcpy 实现拼接
                // 4. 递归调用 traverse_file 函数
            }
        }
    }
    return 0;
}
```

样例输出 &本地测试

对于下发的样例的用户态进程：

```
#include <lib.h>
struct Find_res res __attribute__((aligned(PAGE_SIZE)));

int main() {
    find("/", "motd", &res);
    for (int i = 0; i < res.count; i++) {
        debugf("%s\n", res.file_path[i]);
    }
    return 0;
}
```

其应当输出：

```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
FS is running
superblock is good
read_bitmap is good
/motd
/bin/motd
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
panic at sched.c:45 (schedule): schedule: no runnable envs
```

其中的 `/motd` 和 `/bin/motd` 是找到的文件，两个文件路径输出顺序无所谓。你可以使用：

`make test lab=5_find && make run` 在本地测试上述样例。

提交评测 & 评测标准

请在开发机中执行下列命令后，在**课程网站上提交评测**。

```
$ cd ~/学号/
$ git add -A
$ git commit -m "message" # 请将 message 改为有意义的信息
$ git push
```

测试点说明及分数分布如下：

测试点序号	评测说明	分值
1	**无需实现功能，仅需通过编译**	50 分
2	与样例相同（但文件名不同）	10 分
3	测试数据保证最多有一个满足条件的文件	20 分