Lab6

刘翰文 522030910109

练习 1

在进行 openat 系统调用是，首先为打开的路径分配一个文件描述符用来构建虚拟文件系统文件标识符和真实文件标识符的映射，然后利用路径标识符和当前目录下目标文件名生成完整路径，接着进行 IPC 调用获得真实文件系统内对应文件的路径和挂载信息，最后，利用返回的挂载信息再次进行 IPC 调用进行文件打开操作。

练习题 2

在 fsm_mount_fs 函数内填入以下内容，按照注释进行文件挂载并设定挂载点，创建挂载信息并加入表中，注册 ipc client，申请 ipc 通信：

```
/* Lab 5 TODO Begin (Part 1) */
/* HINT: fsm has the ability to request page_cache syncing and mount and
 * unmount request to the corresponding filesystem. Register an ipc client for each node*/

/* mp_node->_fs_ipc_struct = ipc_register_client(...) */
mp_node->_fs_ipc_struct = ipc_register_client(mp_node->fs_cap);
strlcpy(mp_node->path, mount_point, sizeof(mp_node->path));
/* Increment the fs_num */
fs_num++;
/* Set the correct return value */
ret = 0;
UNUSED(mp_node);

pthread_rwlock_unlock(&mount_point_infos_rwlock);
/* Lab 5 TODO End (Part 1) */
```

练习题 3

在 fsm.c 中的 DEFINE_SERVER_HANDLER 中，填入以下代码，依次完成加锁、获取挂载点信息、获取能力组、设置能力组不存在时对能力组的设立以及设定挂载信息和解锁的操作：

```
/* You should use get_mount_info to get the mount_info and set
 * the fsm_req ipc_msg with mount_id*/

/* lock the mount_info with rdlock */
pthread_rwlock_rdlock(&mount_point_infos_rwlock);
/* mpinfo = get_mount_point(..., ...) */
mpinfo = get_mount_point(fsm_req->path, strlen(fsm_req->path));
/* lock the client_cap_table with mutex */
pthread_mutex_lock(&fsm_client_cap_table_lock);
/* mount_id = fsm_get_client_cap(...) */
mount_id = fsm_get_client_cap(client_badge, mpinfo->fs_cap);
/* if mount_id is not present, we first register the cap set the
 * cap and get mount_id */
if(mount_id == -1){
        mount_id = fsm_set_client_cap(client_badge, mpinfo->fs_cap);
        ret_with_cap = true;
        ipc_set_msg_return_cap_num(ipc_msg, 1);
        ipc_set_msg_cap(ipc_msg, 0, mpinfo->fs_cap);
}
/* set the mount_id, mount_path, mount_path_len in the fsm_req
 */
fsm_req->mount_id = mount_id;
fsm_req->mount_path_len = mpinfo->path_len;
strcpy(fsm_req->mount_path, mpinfo->path);
/* Specifically if we register a new fs_cap in the cap_table, we
 * should let the caller know with a fsm_req->new_cap_flag and
 * then return fs_cap (noted above from mount_id) to the
 * caller*/
fsm_req->new_cap_flag = ret_with_cap;
/* Before returning to the caller , unlock the client_cap_table
 * and mount_info_table */
pthread_mutex_unlock(&fsm_client_cap_table_lock);
pthread_rwlock_unlock(&mount_point_infos_rwlock);
UNUSED(mpinfo);

UNUSED(mount_id);
/* Lab 5 TODO End (Part 1) */
```

练习 4

在 alloc_fs_vnode 函数中，填入以下代码，先后完成对节点的分配、填充对应字段、初始化内容和锁字段即可：

```
/* Lab 5 TODO Begin (Part 2) */
struct fs_vnode *ret = (struct fs_vnode *)malloc(sizeof(*ret));
if(ret==NULL) return NULL;
ret->vnode_id = id;
ret->type = type;
ret->size = size;
ret->private = private;
ret->refcnt = 1;
ret->pmo_cap = -1;
pthread_rwlock_init(&ret->rwlock, NULL);
return ret;

/* Lab 5 TODO End (Part 2) */
```

在 get_fs_vnode_by_id 函数中填入以下代码，利用红黑树查找 id 对应的 vnode：

```
/* Lab 5 TODO Begin (Part 2) */
/* Use the rb_xxx api */
struct rb_node *node = rb_search(fs_vnode_list, &vnode_id, comp_vnode_key);
if(node == NULL) return NULL;
return rb_entry(node, struct fs_vnode, node);
/* Lab 5 TODO End (Part 2) */
```

在 inc_ref_fs_vnode 函数中填入以下代码，将 private node 转换成 fs_vnode 后将 refcnt 增加即可：

```
/* Lab 5 TODO Begin (Part 2) */
/* Private is a fs_vnode */
((struct fs_vnode *)private)->refcnt++;
return 0;
/* Lab 5 TODO End (Part 2) */
```

在 dec_ref_fs_vnode 函数中填入以下代码，在减少 refcnt 外进行是否进行为 0 删除的判断，：

```
/* Lab 5 TODO Begin (Part 2) */
/* Private is a fs_vnode Decrement its refcnt */
int ret;
struct fs_vnode * node = (struct fs_vnode *)private;
node->refcnt--;
if(node->refcnt == 0){
        ret = server_ops.close(node->private, (node->type == FS_NODE_DIR), true);
        if (ret)
                return ret;
        pop_free_fs_vnode(node);
}
return 0;
/* Lab 5 TODO End (Part 2) */
```

练习 5

在 fs_wrapper_set_server_entry 中填入以下代码，对映射不存在的情况添加映射，对正常添加返回 0，对其他情况返回错误代码：

```
/* Lab 5 TODO Begin (Part 3)*/
struct server_entry_node *private_iter;
int ret = 0;
if(fd < 0 || fd >=MAX_SERVER_ENTRY_PER_CLIENT) return -EFAULT;
pthread_spin_lock(&server_entry_mapping_lock);
for_each_in_list(private_iter, struct server_entry_node, node, &server_entry_mapping) {
        if(private_iter->client_badge == client_badge){
                private_iter->fd_to_fid[fd] = fid;
                pthread_spin_unlock(&server_entry_mapping_lock);
                return ret;
        }
}
struct server_entry_node *n = (struct server_entry_node *)malloc(sizeof(*n));
n->client_badge = client_badge;
for(int i = 0; i < MAX_SERVER_ENTRY_PER_CLIENT; i++){
        n->fd_to_fid[i] = -1;
}
n->fd_to_fid[fd] = fid;
list_append(&n->node, &server_entry_mapping);
pthread_spin_unlock(&server_entry_mapping_lock);
return ret;
```

在 fs_wrapper_get_server_entry 中填入以下代码，首先判断是否需要进行转换，接着判断 fd 是否非负且不超过最大值，最后访问映射表找到对应的 client_badge 对应的节点：

```
struct server_entry_node *n;
if(fd == AT_FDROOT) return AT_FDROOT;
if (fd < 0 || fd >= MAX_SERVER_ENTRY_PER_CLIENT) return -1;
pthread_spin_lock(&server_entry_mapping_lock);
for_each_in_list(n, struct server_entry_node, node, &server_entry_mapping)
{
        if(n->client_badge == client_badge){
                pthread_spin_unlock(&server_entry_mapping_lock);
                return n->fd_to_fid[fd];
        }
}
pthread_spin_unlock(&server_entry_mapping_lock);
return -1;
/* Lab 5 TODO End (Part 3)*/
```

练习 6

题目中的函数部分有误，实际需要完成 fs_wrapper_open, fs_wrapper_close, __fs_wrapper_read_core 和__fs_wrapper_writer_core 以及 fs_wrapper_lseek 函数。

在 fs_wrapper_open 函数中填入以下代码，按照注释先获取 fr 中的信息后检查错误码并用 fstatat 判断是否存在，接着用 server_ops.open 操作对文件进行打开操作，根据返回的 vnode_id 查找是否存在 vnode 并以此分别进行增加应用和创建操作，最后建立真正的映射，返回客户端的 fd：

```
int fs_wrapper_open(badge_t client_badge, ipc_msg_t *ipc_msg,
                    struct fs_request *fr)
{
        /* Lab 5 TODO Begin (Part 4)*/
        /* Check the fr permission and open flag if necessary */
        int new_fd = fr->open.new_fd;
        char *path = fr->open.pathname;
        mode_t mode = fr->open.mode;
        int flags = fr->open.flags;
        if((flags & O_CREAT) && (flags & O_EXCL))
        {
                struct stat status;
                if(server_ops.fstatat(path, &status, AT_SYMLINK_NOFOLLOW) == 0)
                        return -EEXIST;
        }
        if((flags & (O_WRONLY | O_RDWR)) && S_ISDIR(mode))
                return -EISDIR;
        if((flags & O_DIRECTORY && !S_ISDIR(mode)))
        {
                return -ENOTDIR;
        }
        /* Use server_ops to open the file */
        ino_t vnode_id;
        off_t vnode_size;
        int vnode_type;
        void *private;
        int ret = server_ops.open(
                path,
                flags,
                mode,
                &vnode_id,
                &vnode_size,
                &vnode_type,
                &private
        );
```

```
if(ret != 0) return -EINVAL;
/* Check if the vnode_id is in rb tree.*/
struct fs_vnode * vnode = get_fs_vnode_by_id(vnode_id);
/* If not, create a new vnode and insert it into the tree. */
if(vnode==NULL){
        vnode = alloc_fs_vnode(vnode_id, vnode_type, vnode_size, private);
        push_fs_vnode(vnode);
}
/* If yes, then close the newly opened vnode and increment the refcnt of
 * present vnode */
else{
        inc_ref_fs_vnode(vnode);
        server_ops.close(private, (vnode_type == FS_NODE_DIR), false);
}
/* Alloc a server_entry and assign the vnode and client generated
 * fd(fr->xxx) to it (Part3 Server fid)*/
int entry_index = alloc_entry();
fr->open.fid = entry_index;
off_t offset = 0;
if((flags & O_APPEND) && S_ISREG(mode))
        offset = vnode_size;
assign_entry(server_entrys[entry_index], flags, offset, 1, (void *)strdup(path), vnode);
fs_wrapper_set_server_entry(client_badge, new_fd, entry_index);
/* Return the client fd */
return new_fd;
/* Lab 5 TODO End (Part 4)*/
```

在 fs_wrapper_close 函数中填入以下代码，在获取当前 fd 指向的 server_ebtry 后将应用递减即可，同时判断是否为 0 并执行相应的删除操作：

```
int fs_wrapper_close(badge_t client_badge, ipc_msg_t *ipc_msg,
                     struct fs_request *fr)
{
        /* Lab 5 TODO Begin (Part 4)*/

        /* Find the server_entry by client fd and client badge */
        struct server_entry * entry = server_entrys[fr->close.fd];
        /* Decrement the server_entry refcnt */
        /* If refcnt is 0, free the server_entry and decrement the vnode
         * refcnt*/
        if(entry->refcnt == 0){
                dec_ref_fs_vnode(entry->vnode);
                fs_wrapper_clear_server_entry(client_badge, fr->close.fd);
                free_entry(fr->close.fd);
        }
        return 0;
        /* Lab 5 TODO End (Part 4)*/
}
```

在__fs_wrapper_read_core 函数中填入以下内容，在判断是否符合权限后进行调用进行读取操作即可：

```
/* Lab 5 TODO Begin (Part 4)*/
/* Use server_ops to read the file into buf. */
/* Do check the boundary of the file and file permission correctly Check
 * Posix Standard for further references. */
if(server_entry->flags & O_WRONLY)
        return -EBADF;
struct fs_vnode * vnode = server_entry->vnode;
ssize_t off = server_ops.read(vnode->private, offset, size, buf);
/* You also should update the offset of the server_entry offset */
return off;
/* Lab 5 TODO End (Part 4)*/
```

在__fs_wrapper_write_core 函数中，填入以下代码，和读取部分的操作类似：

```
/* Lab 5 TODO Begin (Part 4)*/
/* Use server_ops to write the file from buf. */
/* Do check the boundary of the file and file permission correctly Check
 * Posix Standard for further references. */
if((server_entry->flags) & O_RDONLY)
        return -EBADF;
struct fs_vnode * vnode = server_entry->vnode;
ssize_t off = server_ops.write(vnode->private, offset, size, buf);
/* You also should update the offset of the server_entry offset */
return off;
/* Lab 5 TODO End (Part 4)*/
```

在 fs_wrapper_lseek 函数中填入以下代码，按照 whence 不同将游标移到指定的位置：

```
/* Lab 5 TODO Begin (Part 4)*/
/* Check the posix standard. Adjust the server_entry content.*/
off_t offset = fr->lseek.offset;
int fd = fr->lseek.fd;
int whence = fr->lseek.whence;
switch(whence)
{
        case SEEK_SET:
        {
                if(offset < 0) return -EINVAL;
                server_entrys[fd]->offset = offset;
                break;
        }
        case SEEK_CUR:
        {
                if(server_entrys[fd]->offset + offset < 0)
                        return -EINVAL;
                server_entrys[fd]->offset += offset;
                break;
        }
        case SEEK_END:
        {
                if(server_entrys[fd]->vnode->size + offset < 0)
                        return -EINVAL;
                server_entrys[fd]->offset = server_entrys[fd]->vnode->size + offset;
                break;
        }
        default:
                return -EINVAL;
}
ft->lseek.ret = server_entrys[fd]->offset;
return 0;
/* Lab 5 TODO End (Part 4)*/
```