# Report of Lab5

王焕宇 522030910212

## 练习题 1

阅读 user/chcore-libc/libchcore/porting/overrides/src/chcore-port/file.c 的 chcore_openat 函数，分析 ChCore 是如何处理 openat 系统调用的，关注 IPC 的调用过程以及 IPC 请求的内容。

首先 alloc_fd() 分配一个文件描述符，用于构建从虚拟文件系统的文件标识符真实文件系统的文件标识符的映射。
然后，generate_full_path() 利用传入的路径标识符和文件名生成完整的路径。
接着是第一次IPC调用，parse_full_path() 根据文件完整路径获取真实文件系统内对应文件的路径信息和挂载信息。
然后利用请求返回的 mount_id 挂载信息，进行第二次IPC调用，真正进行文件打开操作。
填充进行的ipc请求操作 FS_REQ_OPEN 和进行操作的目标文件在真实文件系统的路径后，进行ipc调用打开文件。

## 练习题 2

实现 user/system-services/system-servers/fsm/fsm.c 的 fsm_mount_fs 函数。

调用 ipc_register_client() 获得对应的ipc通信结构，填充到挂载点对应的ipc通信结构中，函数实现如下：

```
mp_node -> _fs_ipc_struct = ipc_register_client(mp_node -> fs_cap);
strlcpy(mp_node->path, mount_point, sizeof(mp_node->path));
fs_num++;
ret = 0;
pthread_rwlock_unlock(&mount_point_infos_rwlock);
```

# 练习题 3

实现 user/system-services/system-servers/fsm/fsm.c 的 IPC 请求处理函数。

先加锁，后使用 get_mount_point() 获得挂载点信息，然后加互斥锁后进行设定，函数实现如下：

```
pthread_rwlock_rdlock(&mount_point_infos_rwlock);
mpinfo = get_mount_point(fsm_req -> path, strlen(fsm_req -> path));
pthread_mutex_lock(&fsm_client_cap_table_lock);
mount_id = fsm_get_client_cap(client_badge, mpinfo -> fs_cap);
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);
}
fsm_req -> mount_id = mount_id;
fsm_req -> mount_path_len = mpinfo -> path_len;
strcpy(fsm_req -> mount_path, mpinfo -> path);
fsm_req -> new_cap_flag = ret_with_cap;
pthread_mutex_unlock(&fsm_client_cap_table_lock);
pthread_rwlock_unlock(&mount_point_infos_rwlock);
```

# 练习题 4

实现 user/system-services/system-servers/fs_base/fs_vnode.c 中 vnode 的 alloc_fs_vnode、get_fs_vnode_by_id、inc_ref_fs_vnode、dec_ref_fs_vnode 函数。

分配节点，填充对应字段以及初始化锁：

```
struct fs_vnode *alloc_fs_vnode(ino_t id, enum fs_vnode_type type, off_t size,
                                void *private)
{
        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;

}
```

调用 rb_search() 查找 vnode：

```
struct fs_vnode *get_fs_vnode_by_id(ino_t vnode_id)
{
        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);
}
```

增加引用次数：

```
int inc_ref_fs_vnode(void *private)
{
        ((struct fs_vnode *)private)->refcnt++;
        return 0;
}
```

减少引用次数，如果为0则删除并释放资源：

```
int dec_ref_fs_vnode(void *private)
{
        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;
}
```

## 练习题 5

实现 user/system-services/system-servers/fs_base/fs_wrapper.c 中的 fs_wrapper_set_server_entry 和 fs_wrapper_get_server_entry 函数。

遍历映射表，找到对应节点：

```
int fs_wrapper_get_server_entry(badge_t client_badge, int fd)
{
        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;
}
```

分配新的节点加入到节点表中：

```
int fs_wrapper_set_server_entry(badge_t client_badge, int fd, int fid)
{
        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;
}
```

## 练习题 6

实现 user/system-services/system-servers/fs_base/fs_wrapper_ops.c 中的 fs_wrapper_open、fs_wrapper_close、fs_wrapper_read、fs_wrapper_pread、fs_wrapper_write、fs_wrapper_pwrite、fs_wrapper_lseek、fs_wrapper_fmap 函数。

检查文件状态，合法则打开：

```
int fs_wrapper_open(badge_t client_badge, ipc_msg_t *ipc_msg,
                    struct fs_request *fr)
{
        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;
        }
        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;
        struct fs_vnode * vnode = get_fs_vnode_by_id(vnode_id);
        if(vnode == NULL)
        {
                vnode = alloc_fs_vnode(vnode_id, vnode_type, vnode_size, private);
                push_fs_vnode(vnode);
        }
        else
        {
                inc_ref_fs_vnode(vnode);
                server_ops.close(private, (vnode_type == FS_NODE_DIR), false);
        }
```

```
        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 new_fd;
}
```

递减引用数，为0则删除并释放资源：

```
int fs_wrapper_close(badge_t client_badge, ipc_msg_t *ipc_msg,
                     struct fs_request *fr)
{
        struct server_entry * entry = server_entrys[fr->close.fd];
        entry -> 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;
}
```

判断权限，读取：

```
static int __fs_wrapper_read_core(struct server_entry *server_entry, void *buf,
                                  size_t size, off_t offset)
{
        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);
        return off;
}
```

判断权限，写入：

```
static int __fs_wrapper_write_core(struct server_entry *server_entry, void *buf,
                                   size_t size, off_t offset)
{
        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);
        return off;
}


int fs_wrapper_lseek(ipc_msg_t *ipc_msg, struct fs_request *fr)
{
        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;
        }
        fr -> lseek.ret = server_entrys[fd] -> offset;
        return 0;
}
```