```
CS7670, Handout week11b
[https://github.com/sbu-fsl/fuse-stackfs/blob/master/StackFS_LowLevel/StackFS_LowLevel.c]

static void stackfs_ll_open(fuse_req_t req, fuse_ino_t ino,
                    struct fuse_file_info *fi)
{
    int fd;

    generate_start_time(req);
    fd = open(lo_name(req, ino), fi->flags);  // actual work
    generate_end_time(req);
    populate_time(req);

    if (fd == -1)
        return (void) fuse_reply_err(req, errno);

    fi->fh = fd;

    fuse_reply_open(req, fi);
}




static void stackfs_ll_read(fuse_req_t req, fuse_ino_t ino, size_t size,
                off_t offset, struct fuse_file_info *fi)
{
    int res; (void) ino;

    if (USE_SPLICE) {    // with splice optimization
        struct fuse_bufvec buf = FUSE_BUFVEC_INIT(size);

        generate_start_time(req);
        buf.buf[0].flags = FUSE_BUF_IS_FD | FUSE_BUF_FD_SEEK;
        buf.buf[0].fd = fi->fh;
        buf.buf[0].pos = offset;
        generate_end_time(req);
        populate_time(req);

        fuse_reply_data(req, &buf, FUSE_BUF_SPLICE_MOVE);
    } else {  // without splice optimization
        char *buf;

        buf = (char *)malloc(size);
        generate_start_time(req);
        res = pread(fi->fh, buf, size, offset);  // actual work
        generate_end_time(req);
        populate_time(req);
        if (res == -1)
            return (void) fuse_reply_err(req, errno);
        res = fuse_reply_buf(req, buf, res);
        free(buf);
    }
}







static void stackfs_ll_create(fuse_req_t req, fuse_ino_t parent,
        const char *name, mode_t mode, struct fuse_file_info *fi)
{
    int fd, res;
    struct fuse_entry_param e;
    char *fullPath = NULL;
    double attr_val;

    fullPath = (char *)malloc(PATH_MAX);
    construct_full_path(req, parent, fullPath, name);
    attr_val = lo_attr_valid_time(req);
```



```

    generate_start_time(req);

    fd = creat(fullPath, mode);  // acutal creation

    if (fd == -1) {
        if (fullPath)
            free(fullPath);
        generate_end_time(req);
        populate_time(req);
        return (void)fuse_reply_err(req, errno);
    }

    memset(&e, 0, sizeof(e));

    e.attr_timeout = attr_val;
    e.entry_timeout = 1.0;

    res = stat(fullPath, &e.attr);
    generate_end_time(req);
    populate_time(req);

    if (res == 0) {
        /* insert lo_inode into the hash table */
        struct lo_data *lo_data;
        struct lo_inode *lo_inode;

        lo_inode = calloc(1, sizeof(struct lo_inode));
        if (!lo_inode) {
            if (fullPath)
                free(fullPath);

            return (void) fuse_reply_err(req, errno);
        }

        lo_inode->ino = e.attr.st_ino;
        lo_inode->dev = e.attr.st_dev;
        lo_inode->name = strdup(fullPath);
        /* store this for mapping (debugging) */
        lo_inode->lo_ino = (uintptr_t) lo_inode;
        lo_inode->next = lo_inode->prev = NULL;
        free(fullPath);

        lo_data = get_lo_data(req);
        pthread_spin_lock(&lo_data->spinlock);

        res = insert_to_hash_table(lo_data, lo_inode);

        pthread_spin_unlock(&lo_data->spinlock);

        if (res == -1) {
            free(lo_inode->name);
            free(lo_inode);
            fuse_reply_err(req, EBUSY);
        } else {
            lo_inode->nlookup++;
            e.ino = lo_inode->lo_ino;
            //StackFS_trace("Create called, e.ino : %llu", e.ino);
            fi->fh = fd;

            fuse_reply_create(req, &e, fi);
        }
    } else ... // when facing errors
}
```