```c
HANDLER(FUSE_READ)(void *ctx)
{
        struct fuse_req *req = (struct fuse_req *)ctx;
        struct fuse_file_info *fi = req->fi;
        struct lo_inode *inode = get_inode(req);
        size_t count = req->in.h.read.size;
        off_t offset = req->in.h.read.offset;
        char *buf = NULL;
        ssize_t ret = 0;

        if (!inode) {
            ERROR("inode is NULL\n");
            fuse_reply_err(req, EINVAL);
            return;
        }

        // 查找inode的entry条目
        lookup_entry_val_t *entry = bpf_map_lookup_elem(&entry_map, &inode->ino);
        if (!entry || entry->stale) {
            ERROR("entry is NULL or stale\n");
            fuse_reply_err(req, EINVAL);
            return;
        }

        // 在data_map中查找读取的数据
        read_data_key_t key = {
            .nodeid = inode->ino,
        };
        read_data_val_t *val = bpf_map_lookup_elem(&data_map, &key);
        if (val) {
            // 如果在data_map中找到了数据，则直接返回给用户态进程
            fuse_reply_buf(req, val->data, val->size);
            return;
        }

        // 从文件中读取数据
        buf = kmalloc(count, GFP_KERNEL);
        if (!buf) {
            ERROR("kmalloc failed\n");
            fuse_reply_err(req, ENOMEM);
            return;
        }

        ret = kernel_read(fi, buf, count, offset);
        if (ret < 0) {
            ERROR("kernel_read failed\n");
            fuse_reply_err(req, -ret);
            kfree(buf);
                return;
        }

        // 将读取的数据插入到data_map中
        read_data_val_t data = {
            .size = ret,
        };
        memcpy(data.data, buf, ret);
        bpf_map_update_elem(&data_map, &key, &data, BPF_ANY);

        fuse_reply_buf(req, buf, ret);
        kfree(buf);
}
```