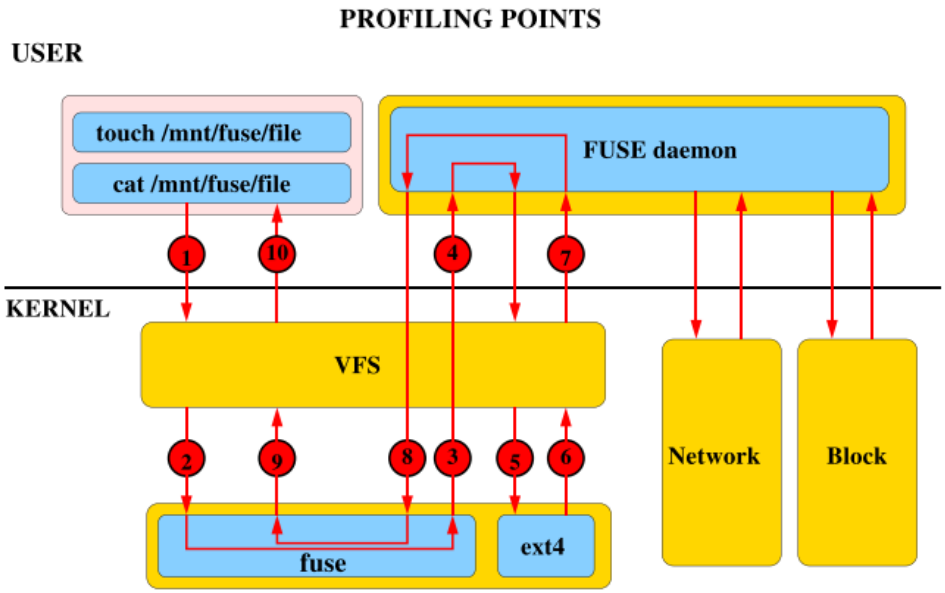
FUSE实现过程中的数据流



"ls -l"命令：

“ls -l”命令进入内核之前：

- 用户空间程序（例如命令行终端）调用`fork()`创建一个子进程。
- 子进程调用`exec()`或其变种函数（如`execve()`）加载并执行`/bin/ls`可执行文件。
- 内核根据可执行文件的路径找到对应的文件，并根据文件格式解析其内容。
- 内核为子进程分配资源，并设置执行环境，然后将控制权转移到`/bin/ls`可执行文件的入口点。
- `/bin/ls`可执行文件运行时使用系统调用`open()`打开当前目录或指定路径。
- 内核根据`open()`的请求，尝试打开指定的目录文件。
- 如果打开成功，`/bin/ls`可执行文件使用系统调用`readdir()`读取该目录的内容，获取文件和子目录列表。
- `/bin/ls`可执行文件将获得的信息输出到终端，完成`ls -l`命令的执行。
- 子进程退出，内核回收相关资源。

"ls -l"命令进入内核后的执行过程（包括getattr）：

- 用户空间的程序发送一个执行命令的请求给操作系统内核。
- 内核虚拟文件系统（VFS）层接收到请求，检查请求的命令类型和路径。
- VFS层判断命令为"ls"，并检查路径是否属于FUSE文件系统。
- 如果路径属于FUSE文件系统，VFS层将该请求转发给FUSE模块对应的用户态进程。
- FUSE进程中的`readdir()`回调函数被调用。
- 在`readdir()`回调函数中，FUSE进程通过与底层实际文件系统的交互，获取目录中的文件和子目录列表。
- FUSE进程将获取到的文件和子目录列表返回给内核模块。
- 内核模块将文件和子目录列表传递给VFS层。
- VFS层将文件和子目录列表返回给用户空间的程序。
- 用户空间的程序解析文件和子目录列表，并按照"-l"选项的要求，对每个文件或目录调用`getattr()`回调函数获取属性信息。
- 在`getattr()`回调函数中，FUSE进程根据请求的路径信息，通过与底层实际文件系统的交互，获取文件或目录的属性信息。
- FUSE进程将获取到的属性信息返回给内核模块。
- 内核模块将属性信息传递给VFS层。

- VFS层将属性信息返回给用户空间的程序。
- 用户空间的程序解析并使用获取到的属性信息。

上述对应阶段中，具体的关键函数如下所示：

- 内核虚拟文件系统（VFS）层接收请求：
  - `sys_open()`：打开文件系统中的文件。
  - `sys_read()`：从文件系统中读取数据。
- 判断命令和路径：
  - `sys_parse_ls_command()`：解析命令并判断是否为"ls"命令。
- 转发请求给FUSE模块对应的用户态进程：
  - `pipe()`：创建管道用于进程间通信。
  - `fork()`：创建子进程用于执行FUSE操作。
- 调用`readdir()`回调函数：
  - `fuse_readdir()`：FUSE进程中的回调函数，用于获取目录中的文件和子目录列表。
- 获取目录中的文件和子目录列表：
  - `readdir()`：FUSE进程中的函数，通过底层实际文件系统的交互获取目录中的文件和子目录列表。
- 返回文件和子目录列表给内核模块：
  - `send_file_list_to_kernel()`：FUSE进程中的函数，将获取到的文件和子目录列表发送给内核模块。
- 内核模块将文件和子目录列表传递给VFS层：
  - `populate_file_list()`：VFS层的函数，接收内核模块传递的文件和子目录列表。
- 返回文件和子目录列表给用户空间的程序：
  - `send_file_list_to_user()`：VFS层的函数，将文件和子目录列表发送给用户空间的程序。
- 解析文件和子目录列表，调用`getattr()`回调函数：
  - 相关解析函数和回调函数：用户空间的程序中的相关函数。
- 获取文件或目录的属性信息：
  - `getattr()`：FUSE进程中的回调函数，通过与底层实际文件系统的交互获取文件或目录的属性信息。
- 返回属性信息给内核模块：
  - `send_attr_to_kernel()`：FUSE进程中的函数，将获取到的属性信息发送给内核模块。
- 内核模块将属性信息传递给VFS层：
  - `set_attr_in_vfs()`：VFS层的函数，接收内核模块传递的属性信息。
- 返回属性信息给用户空间的程序：
  - `send_attr_to_user()`：VFS层的函数，将属性信息发送给用户空间的程序。
- 用户空间的程序解析和使用属性信息

getattr：

在FUSE文件系统中，通常不会直接调用`getattr()`回调函数来处理特定的命令。`getattr()`回调函数更多地用于获取特定文件或目录的属性信息，例如文件大小、权限等。它在许多情况下会被内核或其他FUSE回调函数隐式地调用，而不是由用户空间的程序直接触发。

对于"ls -l"命令，FUSE的`readdir()`回调函数会被调用来获取目录中的文件和子目录列表。然后，对于每个文件或目录，FUSE会自动调用`getattr()`回调函数来获取其属性信息。这是由FUSE的内部机制决定的，用户空间的程序无需显式地调用`getattr()`回调函数。

getattr数据流：

在 FUSE 文件系统中，下面是每个阶段中的一些关键函数：

- 用户请求传递到 VFS 层：用户发起 getattr 请求，请求传递到虚拟文件系统（VFS）层。
  - `do_syscall_64()`：处理系统调用并进行分发
    - `__x64_sys_newstat()`
      - `vfs_statx()`
        - `vfs_getattr()`：根据给定的路径名，获取文件或目录的属性信息

2. VFS 层处理请求后，将其传递给 FUSE 内核：VFS 层将 getattr 请求传递给 FUSE 内核，以便由 FUSE 来处理该请求。
   - fuse_request_send()：将 VFS 请求发送给 FUSE 内核。
3. FUSE 请求分配和初始化完成后，将 FUSE 请求传递给适当的 FUSE 内核队列（此处为 pending 队列）：FUSE 内核接收到 getattr 请求后，将其分配和初始化，并将其放入适当的 FUSE 内核队列中，通常是 pending 队列，等待处理。
   - __fuse_request_alloc()：分配和初始化 FUSE 请求。

```
static struct fuse_req *__fuse_request_alloc(unsigned npages, gfp_t
flags)
{
        struct fuse_req *req = kmem_cache_zalloc(fuse_req_cachep,
flags);
        if (req) {
                struct page **pages = NULL;
                struct fuse_page_desc *page_descs = NULL;

                WARN_ON(npages > FUSE_MAX_MAX_PAGES);
                if (npages > FUSE_REQ_INLINE_PAGES) {
                        pages = fuse_req_pages_alloc(npages, flags,
                                                     &page_descs);
                        if (!pages) {
                                kmem_cache_free(fuse_req_cachep, req);
                                return NULL;
                        }
                } else if (npages) {
                        pages = req->inline_pages;
                        page_descs = req->inline_page_descs;
                }

                fuse_request_init(req, pages, page_descs, npages);
        }
        return req;
}

```

   - fuse_queue_forget()：将请求放入待处理队列。

```
void fuse_queue_forget(struct fuse_conn *fc, struct fuse_forget_link
*forget,
                       u64 nodeid, u64 nlookup)
{
```

```
        struct fuse_iqueue *fiq = &fc->iq;

        forget->forget_one.nodeid = nodeid;
        forget->forget_one.nlookup = nlookup;

        spin_lock(&fiq->waitq.lock);
        if (fiq->connected) {
                fiq->forget_list_tail->next = forget;
                fiq->forget_list_tail = forget;
                wake_up_locked(&fiq->waitq);
                kill_fasync(&fiq->fasync, SIGIO, POLL_IN);
        } else {
                kfree(forget);
        }
        spin_unlock(&fiq->waitq.lock);
}

```

4. FUSE 请求被复制到用户空间守护进程：FUSE 内核将待处理的 getattr 请求复制到用户空间的一个守护进程，该进程用于与用户空间的文件系统实现交互。
    - copy_to_user()：将请求从内核空间复制到用户空间。

```
//  /include/linux/uaccess.h目录下
//  FUSE内核请求数据复制到用户空间时使用此函数，由于其提供了错误处理机制，可以及时检测返回值以处理错误
static __always_inline unsigned long __must_check
copy_to_user(void __user *to, const void *from, unsigned long n)
{
    if (likely(check_copy_size(from, n, true)))
        n = _copy_to_user(to, from, n);
    return n;
}
```

```
//  /tools/virtio/linux/uaccess.h目录下
//  适用于不需要详细错误信息，只关心是否复制成功的场景
static inline int copy_to_user(void __user volatile *to, const void *from,
                   unsigned long n)
{
    __chk_user_ptr(to, n);
    volatile_memcpy(to, from, n);
    return 0;
}
```

5. FUSE 守护进程处理请求并将其传递给底层文件系统（如 Ext4）：FUSE 守护进程接收到 FUSE 内核传递的 getattr 请求后，将其转发给底层的文件系统（如 Ext4）进行处理。
    - fuse_do_getattr()：FUSE 守护进程处理 getattr 请求。

```
static int fuse_do_getattr(struct inode *inode, struct kstat *stat,
                           struct file *file)
```

```
{
        printk(KERN_INFO "fuse_do_getattr\n");
        int err;
        struct fuse_getattr_in inarg;
        struct fuse_attr_out outarg;
        struct fuse_conn *fc = get_fuse_conn(inode);
        FUSE_ARGS(args);
        u64 attr_version;

        attr_version = fuse_get_attr_version(fc);

        memset(&inarg, 0, sizeof(inarg));
        memset(&outarg, 0, sizeof(outarg));
        /* Directories have separate file-handle space */
        if (file && S_ISREG(inode->i_mode)) {
                struct fuse_file *ff = file->private_data;

                inarg.getattr_flags |= FUSE_GETATTR_FH;
                inarg.fh = ff->fh;
        }
        args.in.h.opcode = FUSE_GETATTR;
        args.in.h.nodeid = get_node_id(inode);
        args.in.numargs = 1;
        args.in.args[0].size = sizeof(inarg);
        args.in.args[0].value = &inarg;
        args.out.numargs = 1;
        args.out.args[0].size = sizeof(outarg);
        args.out.args[0].value = &outarg;
        err = fuse_simple_request(fc, &args);
        if (!err) {
                if ((inode->i_mode ^ outarg.attr.mode) & S_IFMT) {
                        make_bad_inode(inode);
                        err = -EIO;
                } else {
                        fuse_change_attributes(inode, &outarg.attr,
                                               attr_timeout(&outarg),
                                               attr_version);
                        if (stat)
                                fuse_fillattr(inode, &outarg.attr,
stat);
                }
        }
        return err;
}
```

    - ext4_getattr()：底层文件系统处理 getattr 请求。
- generic_fillattr()：可以被文件系统自定义的 `getattr()` 函数所调用，用于填充文件的属性信息。

6. Ext4 层执行后，将回复传递给 FUSE 守护进程： Ext4 层处理 getattr 请求后，生成相应的回复，并将回复传递给 FUSE 守护进程。
    - ext4_fill_super()：Ext4 层执行操作并填充回复数据。
7. FUSE 守护进程处理来自 Ext4 层的回复：FUSE 守护进程接收到来自 Ext4 层的回复后，进行处理，可能包括解析回复的数据并执行相应的操作。
    - fuse_reply_entry()：FUSE 守护进程处理并回复 Ext4 层的结果。

8. FUSE 守护进程将回复发送给 FUSE 内核中的待处理队列中的 FUSE 请求，FUSE 内核处理回复：FUSE 守护进程将处理后的回复发送回 FUSE 内核，FUSE 内核接收回复并处理。
    - fuse_request_send()：FUSE 守护进程将回复发送给 FUSE 内核。
9. FUSE 内核将回复发送给 VFS 层，由其进行处理： FUSE 内核将处理后的回复发送回 VFS 层，VFS 层接收到回复并进行处理。
    - fuse_request_send()：FUSE 内核将回复发送给 VFS 层。
10. VFS 层回复用户请求：VFS 层接收到来自 FUSE 内核的回复后，将回复发送给用户空间，完成 getattr 操作的请求处理。
    - fuse_reply_entry()：VFS 层处理并回复用户空间的请求。

read：

“cat”命令：

read数据流：

1. 用户请求传递到VFS层：用户发起read请求，该请求首先传递到虚拟文件系统（VFS）层。
2. VFS层处理请求后，将其传递给FUSE内核：VFS层处理用户请求，并将其传递给FUSE内核，以便在内核空间中进行处理。
3. FUSE请求分配和初始化完成后，将FUSE请求传递给适当的FUSE内核队列：FUSE内核接收到来自VFS层的请求后，将其分配给适当的FUSE内核队列，通常是pending队列。
4. FUSE请求被复制到用户空间守护进程：FUSE内核将请求从内核空间复制到用户空间中运行的FUSE守护进程，以便在用户空间中进行处理。
5. FUSE守护进程处理请求并将其传递给Ext4层：FUSE守护进程在用户空间中处理请求，并将其传递给底层文件系统（例如Ext4）来执行实际的读取操作。
6. Ext4层执行后，将回复传递给FUSE守护进程：底层文件系统（Ext4）执行读取操作，并将读取的数据或错误回复传递回FUSE守护进程。
7. FUSE守护进程处理来自Ext4层的回复：FUSE守护进程接收到来自底层文件系统的回复，并进行相应的处理、转换或错误处理。
8. FUSE守护进程将回复发送给FUSE内核中的待处理队列中的FUSE请求：FUSE守护进程将处理后的回复发送回FUSE内核，并将其放入待处理队列中，等待FUSE内核继续处理。
9. FUSE内核将回复发送给VFS层，由其进行处理：FUSE内核将来自FUSE守护进程的回复发送回VFS层，以便由VFS层进行进一步处理。
10. VFS层回复用户请求：最后，VFS层将处理后的回复发送回用户空间，完成read操作并向用户返回结果。

在这些read操作的执行过程中，涉及到以下关键函数：

1、在VFS层：

- `vfs_read()`：VFS层的文件读取函数，接收用户发起的read请求并将其传递给FUSE内核。

2、在FUSE内核：

- `fuse_request_alloc()`：分配FUSE请求结构体的函数，用于存储用户请求的信息。
- `fuse_copy_args_to_req()`：将用户请求从内核空间复制到FUSE请求结构体中的函数。
- `fuse_queue_forget()`：在处理完请求后，将相关资源（如inode）从FUSE内核中删除的函数。
- `fuse_put_request()`：释放FUSE请求结构体的函数，在处理完请求后将其放回内核的待处理队列中。

3、在用户空间守护进程：

- `fuse_main()`：FUSE守护进程的主要函数，负责启动FUSE文件系统和处理来自FUSE内核的请求。

- `fuse_file_read()`：在FUSE守护进程中用于处理read请求的函数，它调用底层文件系统来执行实际的读取操作。
- `fuse_lowlevel_reply_buf()`：将处理后的回复（读取的数据）发送回FUSE内核的函数。

4、在底层文件系统（例如Ext4）：

- `ext4_file_read_iter()`：底层文件系统中用于执行读取操作的函数，负责从磁盘上读取文件内容。