# ExtFUSE实现过程中的数据流：

ExtFUSE文件系统用户请求的执行过程：

- 用户请求传递到VFS层。
  - 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

- VFS层处理请求后，将其传递给FUSE内核模块。
  - `fuse_request_send()`：VFS层将请求传递给ExtFUSE内核模块的函数。

```
void fuse_request_send(struct fuse_conn *fc, struct fuse_req *req)
{
        if (extfuse_request_send(fc, req) != -ENOSYS)
                return;
        __set_bit(FR_ISREPLY, &req->flags);
        if (!test_bit(FR_WAITING, &req->flags)) {
                __set_bit(FR_WAITING, &req->flags);
                atomic_inc(&fc->num_waiting);
        }
        __fuse_request_send(fc, req);
}
```

- FUSE内核模块经过判断将请求与参数传递给ExtFUSE扩展层
  - `extfuse_request_send()`：/usr/src/linux/fs/fuse/extfuse.c目录中

```
int extfuse_request_send(struct fuse_conn *fc, struct fuse_req *req)
{
    struct extfuse_data *data = (struct extfuse_data *)fc->fc_priv;
    ssize_t ret = -ENOSYS;

    if (data) {
        struct extfuse_req ereq;
        fuse_to_extfuse_req(req, &ereq);
        ret = extfuse_run_prog(data->prog, &ereq);
        if (ret != -ENOSYS) {
            extfuse_to_fuse_req(&ereq, req);
            req->out.h.error = (int)ret;
            ret = 0;
        }
    }

    return ret;
}
```

- `extfuse_to_fuse_req()`：`extfuse_req`结构是扩展的fuse请求结构，可能包含额外的字段或数据。该函数通过选择性地复制并转换字段，将`extfuse_req`结构转换为标准的`fuse_req`结构，以便与fuse驱动程序进行交互。
- `fuse_to_extfuse_req()`：`fuse_req`结构是标准的fuse请求结构，用于与fuse驱动程序进行通信。该函数通过选择性地复制并转换字段，将`fuse_req`结构转换为扩展的`extfuse_req`结构，以便在eBPF程序中进行进一步处理或响应。
- `bpf_extfuse_read_args()`：用于从`extfuse_req`数据结构中将指定字段的值复制到目标容器中。用于从用户空间读取数据到内核空间。

```
BPF_CALL_4(bpf_extfuse_read_args, void *, src, u32, type, void *, dst,
size_t,
       size)
{
    struct extfuse_req *req = (struct extfuse_req *)src;
    unsigned num_in_args = req->in.numargs;
    unsigned num_out_args = req->out.numargs;
    const void *inptr = NULL;
    int ret = -EINVAL;

    switch (type) {
    case OPCODE:
        if (size != sizeof(uint32_t))
            return -EINVAL;
        inptr = (void *)&req->in.h.opcode;
        break;
    case NODEID:
        if (size != sizeof(uint64_t))
            return -EINVAL;
        inptr = (void *)&req->in.h.nodeid;
        break;
    case NUM_IN_ARGS:
        if (size != sizeof(unsigned))
            return -EINVAL;
        inptr = (void *)&req->in.numargs;
        break;
    case NUM_OUT_ARGS:
        if (size != sizeof(unsigned))
            return -EINVAL;
        inptr = (void *)&req->out.numargs;
        break;
    case IN_PARAM_0_SIZE:
        if (size != sizeof(unsigned) || num_in_args < 1 ||
            num_in_args > 3)
            return -EINVAL;
        inptr = &req->in.args[0].size;
        break;
    case IN_PARAM_0_VALUE:
        if (num_in_args < 1 || num_in_args > 3)
            return -EINVAL;
        if (size < req->in.args[0].size)
            return -E2BIG;
        size = req->in.args[0].size;
        inptr = req->in.args[0].value;
        break;
    case IN_PARAM_1_SIZE:
        if (size != sizeof(unsigned) || num_in_args < 2 ||
```

```
            num_in_args > 3)
            return -EINVAL;
        inptr = &req->in.args[1].size;
        break;
    case IN_PARAM_1_VALUE:
        if (num_in_args < 2 || num_in_args > 3)
            return -EINVAL;
        if (size < req->in.args[1].size)
            return -E2BIG;
        size = req->in.args[1].size;
        inptr = req->in.args[1].value;
        break;
    case IN_PARAM_2_SIZE:
        if (size != sizeof(unsigned) || num_in_args != 3)
            return -EINVAL;
        inptr = &req->in.args[2].size;
        break;
    case IN_PARAM_2_VALUE:
        if (num_in_args != 3)
            return -EINVAL;
        if (size < req->in.args[2].size)
            return -E2BIG;
        size = req->in.args[2].size;
        inptr = req->in.args[2].value;
        break;
    case OUT_PARAM_0:
        if (num_out_args < 1 || num_out_args > 2)
            return -EINVAL;
        if (size != req->out.args[0].size)
            return -E2BIG;
        inptr = req->out.args[0].value;
        break;
    case OUT_PARAM_1:
        if (num_out_args != 2)
            return -EINVAL;
        if (size != req->out.args[1].size)
            return -E2BIG;
        inptr = req->out.args[1].value;
        break;
    default:
        return -EBADRQC;
        break;
    }

    if (!inptr) {
        pr_err("Invalid input to %s type: %d num_in_args: %d "
               "num_out_args: %d size: %ld\n",
               __func__, type, num_in_args, num_out_args, size);
        return ret;
    }

    ret = probe_kernel_read(dst, inptr, size);
    if (unlikely(ret < 0))
        memset(dst, 0, size);

    return ret;
}
```

- `bpf_extfuse_write_args()`：用于接收写入操作的参数，并对其进行处理。用于将源字段的内容复制到`extfuse_req`数据结构中的指定输出参数中。用于从内核空间写入数据到用户空间。

```
BPF_CALL_4(bpf_extfuse_write_args, void *, dst, u32, type, const void *,
src,
       u32, size)
{
    struct extfuse_req *req = (struct extfuse_req *)dst;
    unsigned numargs = req->out.numargs;
    void *outptr = NULL;
    int ret = -EINVAL;

    if (type == OUT_PARAM_0 && numargs >= 1 && numargs <= 2 &&
        size == req->out.args[0].size)
        outptr = req->out.args[0].value;

    else if (type == OUT_PARAM_1 && numargs == 2 &&
         size == req->out.args[1].size)
        outptr = req->out.args[1].value;

    if (!outptr) {
        pr_debug("Invalid input to %s type: %d "
             "num_args: %d size: %d\n",
             __func__, type, numargs, size);
        return ret;
    }

    ret = probe_kernel_write(outptr, src, size);
    if (unlikely(ret < 0))
        memset(outptr, 0, size);

    return ret;
}
```

- ExtFUSE扩展层根据请求与请求参数将相应操作下发至ext4文件系统
  - `ext4_readdir()`：
  - `ext4_file_mmap()`：
  - `ext4_filemap_fault()`：
  - `ext4_file_open()`：
  - `ext4_file_read_iter()`：
  - `ext4_file_getattr()`：
    - `ext4_getattr()`：
  - `ext4_release_file()`：
- ExtFUSE 扩展层将结果传递回 FUSE 内核模块
  - `fuse_reply_*()`：这是一组由 FUSE 提供的函数，用于将执行结果打包成 FUSE 协议格式的消息，然后发送给 FUSE 内核模块。具体的函数根据不同的操作类型而有所不同，例如`fuse_reply_attr()`用于返回文件属性，`fuse_reply_open()`用于返回打开文件的结果等。
- FUSE 内核模块将结果传递给 VFS 层

    - `fuse_send_reply()`：这个函数用于将来自用户空间的请求的执行结果发送给 VFS 层。它将执行结果封装成一个响应消息，并将其发送到 VFS 层。
- VFS 层将 ext4 文件系统的执行结果返回给用户空间应用程序
    - `copy_to_user()`：这是 Linux 内核提供的函数，用于将内核空间的数据拷贝到用户空间。在 VFS 层接收到来自 FUSE 内核模块的执行结果后，需要使用 `copy_to_user()` 函数将结果拷贝到用户空间的应用程序提供的缓冲区中。