

## 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。

```
1 void fuse_request_send(struct fuse_conn *fc, struct fuse_req *req)
2 {
3     if (extfuse_request_send(fc, req) != -ENOSYS)
4         return;
5     __set_bit(FR_ISREPLY, &req->flags);
6     if (!test_bit(FR_WAITING, &req->flags)) {
7         __set_bit(FR_WAITING, &req->flags);
8         atomic_inc(&fc->num_waiting);
9     }
10    __fuse_request_send(fc, req);
11 }
```

- 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

```
1 int extfuse_request_send(struct fuse_conn *fc, struct fuse_req *req)
2 {
3     struct extfuse_data *data = (struct extfuse_data *)fc->fc_priv;
4     ssize_t ret = -ENOSYS;
5
6     if (data) {
7         struct extfuse_req ereq;
8         fuse_to_extfuse_req(req, &ereq);
9         ret = extfuse_run_prog(data->prog, &ereq);
10        if (ret != -ENOSYS) {
11            extfuse_to_fuse_req(&ereq, req);
12            req->out.h.error = (int)ret;
13            ret = 0;
14        }
15    }
16
17    return ret;
18 }
```

- `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。

```

1  BPF_CALL_4(bpf_extfuse_read_args, void *, src, u32, type, void *, dst,
2      size_t,
3      size)
4  {
5      struct extfuse_req *req = (struct extfuse_req *)src;
6      unsigned num_in_args = req->in.numargs;
7      unsigned num_out_args = req->out.numargs;
8      const void *inptr = NULL;
9      int ret = -EINVAL;
10
11     switch (type) {
12     case OPCODE:
13         if (size != sizeof(uint32_t))
14             return -EINVAL;
15         inptr = (void *)&req->in.h.opcode;
16         break;
17     case NODEID:
18         if (size != sizeof(uint64_t))
19             return -EINVAL;
20         inptr = (void *)&req->in.h.nodeid;
21         break;
22     case NUM_IN_ARGS:
23         if (size != sizeof(unsigned))
24             return -EINVAL;
25         inptr = (void *)&req->in.numargs;
26         break;
27     case NUM_OUT_ARGS:
28         if (size != sizeof(unsigned))
29             return -EINVAL;
30         inptr = (void *)&req->out.numargs;
31         break;
32     case IN_PARAM_0_SIZE:
33         if (size != sizeof(unsigned) || num_in_args < 1 ||
34             num_in_args > 3)
35             return -EINVAL;
36         inptr = &req->in.args[0].size;
37         break;
38     case IN_PARAM_0_VALUE:
39         if (num_in_args < 1 || num_in_args > 3)
40             return -EINVAL;
41         if (size < req->in.args[0].size)
42             return -E2BIG;
43         size = req->in.args[0].size;
44         inptr = req->in.args[0].value;
45         break;
46     case IN_PARAM_1_SIZE:
47         if (size != sizeof(unsigned) || num_in_args < 2 ||

```

```

47         num_in_args > 3)
48         return -EINVAL;
49     inptr = &req->in.args[1].size;
50     break;
51 case IN_PARAM_1_VALUE:
52     if (num_in_args < 2 || num_in_args > 3)
53         return -EINVAL;
54     if (size < req->in.args[1].size)
55         return -E2BIG;
56     size = req->in.args[1].size;
57     inptr = req->in.args[1].value;
58     break;
59 case IN_PARAM_2_SIZE:
60     if (size != sizeof(unsigned) || num_in_args != 3)
61         return -EINVAL;
62     inptr = &req->in.args[2].size;
63     break;
64 case IN_PARAM_2_VALUE:
65     if (num_in_args != 3)
66         return -EINVAL;
67     if (size < req->in.args[2].size)
68         return -E2BIG;
69     size = req->in.args[2].size;
70     inptr = req->in.args[2].value;
71     break;
72 case OUT_PARAM_0:
73     if (num_out_args < 1 || num_out_args > 2)
74         return -EINVAL;
75     if (size != req->out.args[0].size)
76         return -E2BIG;
77     inptr = req->out.args[0].value;
78     break;
79 case OUT_PARAM_1:
80     if (num_out_args != 2)
81         return -EINVAL;
82     if (size != req->out.args[1].size)
83         return -E2BIG;
84     inptr = req->out.args[1].value;
85     break;
86 default:
87     return -EBADRQC;
88     break;
89 }
90
91 if (!inptr) {
92     pr_err("Invalid input to %s type: %d num_in_args: %d "
93           "num_out_args: %d size: %ld\n",
94           __func__, type, num_in_args, num_out_args, size);
95     return ret;
96 }
97
98 ret = probe_kernel_read(dst, inptr, size);
99 if (unlikely(ret < 0))
100     memset(dst, 0, size);
101
102 return ret;
103 }

```

- `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。

```

1 BPF_CALL_4(bpf_extfuse_write_args, void *, dst, u32, type, const void *,
2           src,
3           u32, size)
4 {
5     struct extfuse_req *req = (struct extfuse_req *)dst;
6     unsigned numargs = req->out.numargs;
7     void *outptr = NULL;
8     int ret = -EINVAL;
9
10    if (type == OUT_PARAM_0 && numargs >= 1 && numargs <= 2 &&
11        size == req->out.args[0].size)
12        outptr = req->out.args[0].value;
13
14    else if (type == OUT_PARAM_1 && numargs == 2 &&
15             size == req->out.args[1].size)
16        outptr = req->out.args[1].value;
17
18    if (!outptr) {
19        pr_debug("Invalid input to %s type: %d "
20                "num_args: %d size: %d\n",
21                __func__, type, numargs, size);
22        return ret;
23    }
24
25    ret = probe_kernel_write(outptr, src, size);
26    if (unlikely(ret < 0))
27        memset(outptr, 0, size);
28
29    return ret;
30 }

```

- 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。