**efuse\_comp.h**

#define EFUSE\_MAX\_COMP 1 *// Maximum number of completion threads*

#define EFUSE\_COMP\_MAX\_IDLE 50 *// Maximum idle time of completion thread*

#define EFUSE\_WAKE\_UP\_COMP 37

#define EFUSE\_REPLY\_ASYNC 38

#define EFUSE\_DAEMON\_SLEEP 39

#define COMP\_NEED\_WAKEUP (1U << 0)

#define COMP\_SLEEPING (1U << 1)

#define COMP\_WORKING (1U << 2)

#define COMP\_NEED\_WAKEUP\_FROM\_USER (1U << 3)

#define COMP\_ALL\_COMP\_WORKING (1U << 4)

**功能**：

定义了与完成线程相关的常量：

EFUSE\_MAX\_COMP：最大完成线程数。

EFUSE\_COMP\_MAX\_IDLE：完成线程的最大空闲时间（单位为微秒）。

EFUSE\_WAKE\_UP\_COMP、EFUSE\_REPLY\_ASYNC、EFUSE\_DAEMON\_SLEEP：与完成线程相关的状态码。

定义了完成线程的状态标志：

COMP\_NEED\_WAKEUP：需要唤醒。

COMP\_SLEEPING：正在睡眠。

COMP\_WORKING：正在工作。

COMP\_NEED\_WAKEUP\_FROM\_USER：由用户请求唤醒。

COMP\_ALL\_COMP\_WORKING：所有完成线程都在工作。

**区别**：

传统 FUSE 没有专门的完成线程机制，而 Efuse 引入了完成线程来异步处理请求完成事件。

Efuse 的完成线程机制提供了更细粒度的控制和状态管理。

struct efuse\_comp\_entry {

int comp\_id;

u32 comp\_state;

struct task\_struct \*comp\_kthread;

struct list\_head list;

};

struct efuse\_comp\_head {

spinlock\_t lock;

struct list\_head list;

};

struct efuse\_comp\_args {

struct fuse\_conn \*fc;

int riq\_id;

int comp\_id;

struct efuse\_comp\_entry \*entry;

};

**功能**：

**efuse\_comp\_entry**：

表示一个完成线程的条目。

包含完成线程的 ID（comp\_id）、状态（comp\_state）和线程结构（comp\_kthread）。

使用链表（list\_head）来管理完成线程。

**efuse\_comp\_head**：

管理所有完成线程的头部结构。

包含一个锁（spinlock\_t lock）和一个链表（list\_head list）。

**efuse\_comp\_args**：

用于传递完成线程的参数。

包含 FUSE 连接（fc）、队列 ID（riq\_id）、完成线程 ID（comp\_id）和完成线程条目（entry）。

**区别**：

传统 FUSE 没有专门的完成线程管理结构，而 Efuse 引入了这些结构来支持异步完成处理。

Efuse 的完成线程机制允许更灵活地管理完成事件，提高系统的并发处理能力。

void efuse\_sleep\_comp(struct fuse\_conn \*fc, struct efuse\_iqueue \*riq, struct efuse\_req \*r\_req) {

spin\_lock(&r\_req->waitq.lock);

set\_bit(FR\_NEEDWAKEUP, &r\_req->flags);

spin\_unlock(&r\_req->waitq.lock);

spin\_lock(&riq->lock);

riq->num\_sync\_sleeping++;

spin\_unlock(&riq->lock);

wait\_event\_interruptible(r\_req->waitq, !fc->connected || test\_bit(FR\_FINISHED, &r\_req->flags));

spin\_lock(&riq->lock);

riq->num\_sync\_sleeping--;

spin\_unlock(&riq->lock);

}

**功能**：

将请求标志设置为需要唤醒（FR\_NEEDWAKEUP）。

增加队列中同步睡眠的请求数量（num\_sync\_sleeping）。

使请求进入睡眠状态，等待完成或连接断开。

减少队列中同步睡眠的请求数量。

**区别**：

传统 FUSE 没有专门的睡眠机制，而 Efuse 提供了更复杂的睡眠和唤醒机制。Efuse 的睡眠机制允许请求在等待完成时进入睡眠状态，减少 CPU 使用率。

int efuse\_completion\_poll(struct fuse\_conn \*fc, struct efuse\_iqueue \*riq, struct efuse\_req \*r\_req)

{

unsigned long max\_idle\_due = jiffies + usecs\_to\_jiffies(EFUSE\_COMP\_MAX\_IDLE);

while(fc->connected) {

if(test\_bit(FR\_FINISHED, &r\_req->flags)){

efuse\_request\_end(r\_req);

return 0;

}

if(time\_after(jiffies, max\_idle\_due)){

efuse\_sleep\_comp(fc, riq, r\_req);

}

schedule();

}

return -ENOTCONN;

}

**功能**：

计算最大空闲时间（max\_idle\_due）。

在连接有效的情况下，轮询请求是否完成：

如果请求完成（FR\_FINISHED），调用 efuse\_request\_end 处理完成事件并返回。

如果超过最大空闲时间，调用 efuse\_sleep\_comp 使请求进入睡眠状态。

如果连接断开，返回 -ENOTCONN。

**区别**：

传统 FUSE 没有专门的轮询机制，而 Efuse 提供了更复杂的轮询和睡眠机制。Efuse 的轮询机制允许请求在等待完成时进入睡眠状态，减少 CPU 使用率。Efuse 的轮询机制支持最大空闲时间的设置，进一步优化性能。