**Efuse.h**

#define EFUSE\_NUM\_IQUEUE 12           // Number of efuse iqueue

#define EFUSE\_MAX\_QUEUE\_SIZE 1024 \* 4 // Maximum number of requests in a queue

#define EFUSE\_RIQ\_ID\_MASK 0x00ff0000ULL

#define EFUSE\_QUEUE\_MAP\_MASK 0xff000000ULL

#define EFUSE\_REQ\_IDX\_MASK 0x0000ffff00000000ULL

#define EFUSE\_IQUEUE 0ULL

#define EFUSE\_PENDING 0x08000000ULL

#define EFUSE\_INTERRUPT 0x18000000ULL

#define EFUSE\_FORGET 0x20000000ULL

#define EFUSE\_COMPLETE 0x28000000ULL

#define EFUSE\_ARG 0x30000000ULL

#define EFUSE\_REQ 0x38000000ULL

#define EFUSE\_READ 0x40000000ULL

#define EFUSE\_WRITE 0x48000000ULL

功能

定义了 Efuse 的队列数量（EFUSE\_NUM\_IQUEUE）和每个队列的最大请求数量（EFUSE\_MAX\_QUEUE\_SIZE）。

定义了用于标识请求队列类型和请求索引的掩码（EFUSE\_RIQ\_ID\_MASK、EFUSE\_QUEUE\_MAP\_MASK、EFUSE\_REQ\_IDX\_MASK）。

定义了不同类型的队列标识符（如 EFUSE\_IQUEUE、EFUSE\_PENDING、EFUSE\_INTERRUPT 等）。

区别

传统 FUSE 通常只有一个请求队列，而 Efuse 引入了多个队列（如挂起队列、中断队列、遗忘队列和完成队列），以支持更复杂的请求管理。

Efuse 使用掩码和标识符来区分不同类型的队列，这在传统 FUSE 中是不存在的。

struct efuse\_req {

*/\*\* Request input header \*\*/*

struct {

uint64\_t unique;

uint64\_t nodeid;

uint32\_t opcode;

uint32\_t uid;

uint32\_t gid;

uint32\_t pid;

uint32\_t arg[2]; *// Location of in operation-specific argument*

uint32\_t arglen[2]; *// Size of in operation-specific argument*

} in; *// 48*

*/\*\* Request output header \*\*/*

struct {

int32\_t error;

uint32\_t arg; *// Location of out operation-specific argument*

uint32\_t arglen; *// Size of out operation-specific argument*

uint32\_t padding;

} out; *// 16*

*/\*\* request buffer index \*\*/*

uint32\_t index; *// 4*

int32\_t riq\_id;

*/\*\* Request flags, updated with test/set/clear\_bit() \*\*/*

unsigned long flags; *// 8*

*/\*\* fuse\_mount this request belongs to \*\*/*

struct fuse\_mount \*fm; *// 8*

*/\*\* refcount \*\*/*

refcount\_t count; *// 4*

*/\*\* Used to wake up the task waiting for completion of request \*\*/*

wait\_queue\_head\_t waitq; *// 24*

struct {

uint8\_t argument\_space[112];

} args; *// 112*

bool force : 1;

bool noreply : 1;

bool nocreds : 1;

bool in\_pages : 1;

bool out\_pages : 1;

bool out\_argvar : 1;

bool page\_zeroing : 1;

bool page\_replace : 1;

bool may\_block : 1;

struct efuse\_pages \*rp;

void (\*end)(struct fuse\_mount \*fm, struct efuse\_req \*r\_req, int error);

};

功能

定义了 Efuse 请求的结构，包含请求的输入头（in）、输出头（out）、请求缓冲区索引（index）、队列 ID（riq\_id）、请求标志（flags）、引用计数（count）、等待队列（waitq）等。

包含了请求的参数空间（args）和布尔标志位（如 force、noreply、nocreds 等）。

包含了回调函数指针（end），用于处理请求完成后的操作。

区别

传统 FUSE 的请求结构相对简单，而 Efuse 的请求结构更加复杂，增加了更多的字段和标志位，以支持更复杂的请求管理和异步操作。

Efuse 引入了引用计数（refcount\_t count）和等待队列（wait\_queue\_head\_t waitq），用于更好地管理请求的生命周期和同步操作。

struct efuse\_interrupt\_entry {

  uint64\_t unique;

};

struct efuse\_forget\_entry {

  uint64\_t unique;

  uint64\_t nlookup;

  uint64\_t nodeid;

  uint64\_t padding;

};

struct efuse\_address\_entry {

  uint32\_t request;

}; // 4 bytes

struct efuse\_bg\_entry {

  struct list\_head list;

  uint32\_t request;

  int32\_t riq\_id;

};

功能

定义了中断队列的条目结构，包含一个唯一的请求标识符（unique）。

定义了遗忘队列的条目结构，包含唯一的请求标识符（unique）、查找次数（nlookup）和节点 ID（nodeid）。

定义了请求地址条目结构，包含请求的索引（request）

定义了后台队列的条目结构，包含请求的索引（request）和队列 ID（riq\_id）。

使用链表（list\_head）来管理后台队列。

区别

传统 FUSE 没有专门的中断队列，而 Efuse 引入了中断队列，用于处理请求的中断操作。

传统 FUSE 也有遗忘队列，但 Efuse 的遗忘队列结构更加详细，增加了更多的字段

传统 FUSE 没有类似的结构，Efuse 引入了请求地址条目，用于管理请求的地址信息

传统 FUSE 没有后台队列的概念，而 Efuse 引入了后台队列，用于管理后台请求。

struct ring\_buffer\_1 {

uint32\_t tail;

uint32\_t head;

uint32\_t mask;

uint32\_t entries;

struct efuse\_address\_entry \*kaddr; *// kernel address*

struct efuse\_address\_entry \*uaddr; *// user address*

};

struct ring\_buffer\_2 {

uint32\_t head;

uint32\_t tail;

uint32\_t mask;

uint32\_t entries;

struct efuse\_interrupt\_entry \*kaddr; *// kernel address*

struct efuse\_interrupt\_entry \*uaddr; *// user address*

};

struct ring\_buffer\_3 {

uint32\_t head;

uint32\_t tail;

uint32\_t mask;

uint32\_t entries;

struct efuse\_forget\_entry \*kaddr; *// kernel address*

struct efuse\_forget\_entry \*uaddr; *// user address*

};

功能

定义了三种环形缓冲区结构，分别用于管理挂起队列（ring\_buffer\_1）、中断队列（ring\_buffer\_2）和遗忘队列（ring\_buffer\_3）。

每个环形缓冲区包含头指针（head）、尾指针（tail）、掩码（mask）和条目数量（entries）。

包含内核地址（kaddr）和用户地址（uaddr）。

区别

Efuse 引入了多个专门的队列，包括挂起队列、中断队列、忘记队列和完成队列。每个队列都有特定的用途。

这种多队列机制可以提高系统的并发处理能力和性能，因为不同类型的操作可以并行处理，避免了单一队列中的阻塞。

Efuse 的环形缓冲区支持内核空间和用户空间的共享，通过 kaddr 和 uaddr 分别表示内核地址和用户地址。

这种设计允许用户空间直接访问和操作缓冲区，减少了内核和用户空间之间的数据拷贝，提高了性能。

struct efuse\_iqueue {

int riq\_id;

*/\*\* Pending queue \*\*/*

struct ring\_buffer\_1 pending;

*/\*\* Interrupt queue \*\*/*

struct ring\_buffer\_2 interrupts;

*/\*\* Forget queue \*\*/*

struct ring\_buffer\_3 forgets;

*/\*\* Complete queue \*\*/*

struct ring\_buffer\_1 completes;

*/\*\* Dyanmic argument buffer \*\*/*

struct efuse\_arg \*uarg; *// user address*

struct efuse\_arg \*karg; *// kernel address*

*/\*\* Dynamic request buffer \*\*/*

struct efuse\_req \*ureq; *// user address*

struct efuse\_req \*kreq; *// kernel address*

*/\*\* Connection established \*\*/*

unsigned connected;

*/\*\* wait queue for requests to wait to receive a request buffer \*\*/*

wait\_queue\_head\_t waitq;

*/\*\* Lock protecting accesses to members of this structure \*\*/*

spinlock\_t lock;

*/\*\* The next unique request id \*\*/*

u64 reqctr;

*/\*\* Device specific state \*/*

void \*priv;

struct {

unsigned long bitmap\_size;

unsigned full;

unsigned long \*bitmap;

} argbm;

struct {

unsigned long bitmap\_size;

unsigned full;

unsigned long \*bitmap;

} reqbm;

wait\_queue\_head\_t idle\_user\_waitq;

*/\*\* synchronous request congestion control \*/*

int num\_sync\_sleeping;

*/\*\* background request congestion control \*/*

struct list\_head bg\_queue;

spinlock\_t bg\_lock;

unsigned max\_background;

unsigned congestion\_threshold;

unsigned num\_background;

unsigned active\_background;

int blocked;

*/\*\* waitq for congested asynchronous requests\*/*

wait\_queue\_head\_t blocked\_waitq;

};

**功能**：

定义了 Efuse 的请求队列结构（efuse\_iqueue），用于管理不同类型的请求队列。

包含挂起队列（pending）、中断队列（interrupts）、忘记队列（forgets）和完成队列（completes）。

包含动态参数缓冲区（uarg 和 karg）和动态请求缓冲区（ureq 和 kreq）。

包含连接状态（connected）、等待队列（waitq）、锁（lock）、请求计数器（reqctr）和设备特定状态（priv）。

包含参数和请求缓冲区的位图管理（argbm 和 reqbm）。

包含同步请求和后台请求的拥塞控制机制（num\_sync\_sleeping、bg\_queue、bg\_lock、max\_background、congestion\_threshold、num\_background、active\_background、blocked 和 blocked\_waitq）。

**区别**：

传统 FUSE 通常只有一个请求队列，而 Efuse 引入了多个队列，以支持更复杂的请求管理。

Efuse 使用位图来管理参数和请求缓冲区的分配和释放，这在传统 FUSE 中是不存在的。

Efuse 引入了同步请求和后台请求的拥塞控制机制，以优化性能和资源管理。