Linux 内核 IDR 机制

Author:jeffrey.lau.td

Date:2012

idr 在 linux 内核中指的就是整数 ID 管理机制,从本质上来说,这就是一种将整数 ID 号和特定指针关联在一起的机制.idr 机制适用在那些需要把某个整数和特定指针关联在一起的地方.举个例子,在 I2C 总线中,每个设备都有自己的地址,要想在总线上找到特定的设备,就必须要先发送该设备的地址.如果我们的 PC 是一个 I2C 总线上的主节点,那么要访问总线上的其他设备,首先要知道他们的 ID 号,同时要在 pc 的驱动程序中建 立一个用于描述该设备的结构体.此时,问题来了,我们怎么才能将这个设备的 ID 号和他的设备结构体联系起来呢？最简单的方法当然是通过数组进行索引,但如果 ID 号的范围很大(比如 32 位 的 ID 号),则用数组索引显然不可能; 第二种方法是用链表,但如果网络中实际存在的设备较多,则链表的查询效率会很低.遇到这种清况,我们就可以采用 idr 机制,该机制内部采用 radix 树实现,可以很方便地将整数和指针关联起来,并且具有很高的搜索效率

idr 数据结构: linux/idr.h

```
struct idr {
    struct idr_layer __rcu *top;
    struct idr_layer *id_free;
    int  layers; /*only valid without concurrent changes */
    int        id_free_cnt;
    spinlock_t     lock;
};
```

1)idr 静态声明并初始化:

```
#define IDR_INIT(name)                                  \
{                                       \
    .top     = NULL,                        \
    .id_free = NULL,                        \
    .layers  = 0,                       \
    .id_free_cnt = 0,                       \
    .lock        = __SPIN_LOCK_UNLOCKED(name.lock), \
}
#define DEFINE_IDR(name)  struct idr name = IDR_INIT(name)
static DEFINE_IDR(i2c_adapter_idr);
```

2)idr 动态初始化:

```
/**
 * idr_init - initialize idr handle
 * @idp: idr handle
 * This function is use to set up the handle (@idp) that
     you will pass
 * to the rest of the functions.
```

```
 */
void idr_init(struct idr *idp)
{
     memset(idp, 0, sizeof(struct idr));
     spin_lock_init(&idp->lock);
}
static struct idr i2c_adapter_idr;
idr_init(&i2c_adapter_idr);
```

3)为 idr 分配内存

```
/**
 * idr_pre_get - reserve resources for idr allocation
 * @idp: idr handle
 * @gfp_mask: memory allocation flags
 * 每次通过 idr 获得 ID 号之前,需要先分配内存。
 * 返回 0 表示错误,非零值代表正常
 */
int idr_pre_get(struct idr *idp, gfp_t gfp_mask)
{
     while (idp->id_free_cnt < IDR_FREE_MAX) {
          struct idr_layer *new;
          new = kmem_cache_zalloc(idr_layer_cache,
                                              gfp_mask);
          if (new == NULL)
               return (0);
          move_to_free_list(idp, new);
     }
     return 1;
}
if (idr_pre_get(&i2c_adapter_idr, GFP_KERNEL) == 0)
          return -ENOMEM;
```

4)分配 ID 号并将 ID 号和设备指针关联起来

将 ptr 结构插入到 idr 中

```
/**
 * idr_get_new_above - allocate new idr entry above or
     equal to a start id
 * @idp: idr handle
 * @ptr: pointer you want associated with the id
 * @starting_id: id to start search at
 * @id: pointer to the allocated handle
 * 如果成功 return 0 如果无 ID 分配 return -ENOSPC
 */
int idr_get_new_above(struct idr *idp, void *ptr,
                              int starting_id, int *id)
```

```
int idr_get_new(struct idr *idp, void *ptr, int *id)
```

5)通过 ID 号搜索对应的指针

```
void *idr_find(struct idr *idp, int id);
```

返回值是和给定 id 相关联的指针,如果没有,则返回 NULL

6)删除 ID

```
void idr_remove(struct idr *idp, int id);
void idr_remove_all(struct idr *idp);
```

附件代码: i2c_add_adapter 的实现

```
#include <linux/idr.h>

static DEFINE_IDR(i2c_adapter_idr);

/**
 * i2c_add_adapter - declare i2c adapter, use dynamic bus
     number
 * @adapter: the adapter to add
 * Context: can sleep
 * 动态注册一个 adapter
 */
int i2c_add_adapter(struct i2c_adapter *adapter)
{
    int  id, res = 0;

retry:
    if (idr_pre_get(&i2c_adapter_idr, GFP_KERNEL) == 0)
        return -ENOMEM;

    mutex_lock(&core_lock);
    /* "above" here means "above or equal to", sigh */
    /*将 adapter 结构插入到 i2c_adapter_idr 当中,并且存放的位置必须
      是大于或者等于__i2c_first_dynamic_bus_num
    */
    res = idr_get_new_above(&i2c_adapter_idr, adapter,
                __i2c_first_dynamic_bus_num, &id);
    mutex_unlock(&core_lock);

    if (res < 0) {
        if (res == -EAGAIN)
            goto retry;
        return res;
    }
    /*这里将对应的 ID 号存放到 adapter->nr*/
    adapter->nr = id;
```

```
    //最后回调注册到 I2C-CORE
    return i2c_register_adapter(adapter);
}
```

```
/**
 * i2c_add_numbered_adapter - declare i2c adapter, use
static bus number
 * @adap: the adapter to register (with adap->nr
initialized)
 * Context: can sleep
 *静态添加 adapter
 */
int i2c_add_numbered_adapter(struct i2c_adapter *adap)
{
    int  id;
    int  status;

    if (adap->nr & ~MAX_ID_MASK)
        return -EINVAL;

retry:
    if (idr_pre_get(&i2c_adapter_idr, GFP_KERNEL) == 0)
        return -ENOMEM;

    mutex_lock(&core_lock);
    /* "above" here means "above or equal to", sigh;
     * we need the "equal to" result to force the result
     */
    status = idr_get_new_above(&i2c_adapter_idr,
                               adap, adap->nr, &id);
    /*如果分配的 ID 号和 adap->nr 不匹配则返回出错*/
    if (status == 0 && id != adap->nr) {
        status = -EBUSY;
        idr_remove(&i2c_adapter_idr, id);
    }
    mutex_unlock(&core_lock);
    if (status == -EAGAIN)
        goto retry;

    if (status == 0)
        status = i2c_register_adapter(adap);
    return status;
}
```

```
/**
 * i2c_del_adapter - unregister I2C adapter
```

```
 * @adap: the adapter being unregistered
 * Context: can sleep
 *
 */
int i2c_del_adapter(struct i2c_adapter *adap)
{

    /* First make sure that this adapter was ever added */
    mutex_lock(&core_lock);
    //先找到对应的 ID 号
    found = idr_find(&i2c_adapter_idr, adap->nr);
    mutex_unlock(&core_lock);

    /* free bus id */
    mutex_lock(&core_lock);
    //将 adap->nr 对应的指针从 i2c_adapter_idr 删除掉
    idr_remove(&i2c_adapter_idr, adap->nr);
    mutex_unlock(&core_lock);


    return 0;
}
```