第 0019 讲 10kmalloc 案例实战分析

一、kmalloc/kfree 函数原型

1、kmalloc()函数功能：该函数是内核中动态内存分配函数之一，主要用于从内核空间中分配连续的物理内存块。

2、kmalloc()函数原型

include > linux > C slab.h > ...

```
543 static __always_inline void *kmalloc(size_t size, gfp_t flags)
544 {
545     if (__builtin_constant_p(size)) {
546 #ifndef CONFIG_SLOB
547         unsigned int index;
548 #endif
549         if (size > KMALLOC_MAX_CACHE_SIZE)
550             return kmalloc_large(size, flags);
551 #ifndef CONFIG_SLOB
552         index = kmalloc_index(size);
553 
554         if (!index)
555             return ZERO_SIZE_PTR;
556 
557         return kmem_cache_alloc_trace(
558                 kmalloc_caches[kmalloc_type(flags)][index],
559                 flags, size);
560 #endif
561     }
562     return __kmalloc(size, flags);
563 }
```

- size：是指要分配的内存的字节数。
- flags：是分配标志，提供多种 kmalloc( )的行为。

该函数返回值：返回一个指向分配的内存块起始地址的地址指针。

```
static __always_inline void *kmalloc(size_t size, gfp_t flags)
```

```
{
    // 内核用此函数来优化代码，编译期间通过静态分析装饰一些条件判断和分支转换为
常量
    if (__builtin_constant_p(size)) {
#ifndef CONFIG_SLOB
        unsigned int index;
#endif
        // 小于等于 KMALLOC_MAX_CACHE_SIZE 时，使用预先定义好的内存缓存进行分
配
        // ddgf KMALLOC_MAX_CACHE_SIZE 时调用 kmalloc_large 进行大块内存的申
请
        if (size > KMALLOC_MAX_CACHE_SIZE)
            return kmalloc_large(size, flags);
#ifndef CONFIG_SLOB
        // kmalloc_index 该函数主要用于计算要分配内存大小对应的缓存池索引
        index = kmalloc_index(size);
        if (!index)
            return ZERO_SIZE_PTR;
        // kmalloc_type：数据则记录预先定义好的一系列缓存池
        return kmem_cache_alloc_trace(
                kmalloc_caches[kmalloc_type(flags)][index],
                flags, size); // 从 slab 分配内存（分配 slab 缓存）
#endif
    }

    // __kmalloc 该函数用于从内存池（slab）中分配指定大小的内存块，
```

```
    // 返回分配到的内存块地址

    return __kmalloc(size, flags);
}
```

3、kfree()函数原型

```
mm > C slab.c > ...
3741  void kfree(const void *objp)
3742  {
3743      struct kmem_cache *c;
3744      unsigned long flags;
3745
3746      trace_kfree(_RET_IP_, objp);
3747
3748      if (unlikely(ZERO_OR_NULL_PTR(objp)))
3749          return;
3750      local_irq_save(flags);
3751      kfree_debugcheck(objp);
3752      c = virt_to_cache(objp);
3753      if (!c) {
3754          local_irq_restore(flags);
3755          return;
3756      }
3757      debug_check_no_locks_freed(objp, c->object_size);
3758
3759      debug_check_no_obj_freed(objp, c->object_size);
3760      __cache_free(c, (void *)objp, _RET_IP_);
3761      local_irq_restore(flags);
3762  }
3763  EXPORT_SYMBOL(kfree);
3764
```

主要作用：释放地址 objp 开始的一段内存。

```
void kfree(const void *objp)
{

    struct kmem_cache *c;

    unsigned long flags;

    // 调用该函数 trace_kfree 释放之前从内存池中分配到的指定大小的内存池，并追踪相关信息
```

```
    trace_kfree(_RET_IP_, objp);

    // 调用该函数 unlikely 检查等待释放的内存块指针是否为空或者为零（NULL）
    if (unlikely(ZERO_OR_NULL_PTR(objp)))
        return;

    // 调用该函数，先将当前 CPU 的标志寄存器值保存在 flags 变量当中
    local_irq_save(flags);
```

```
    // 用于释放内存的函数
    kfree_debugcheck(objp);
```

```
    // 用于获取虚拟地址所映射的缓存页描述符的函数
    c = virt_to_cache(objp);
    if (!c) {
        local_irq_restore(flags);
        return;
    }
    debug_check_no_locks_freed(objp, c->object_size);
    debug_check_no_obj_freed(objp, c->object_size);
    __cache_free(c, (void *)objp, _RET_IP_);
    local_irq_restore(flags);
}
EXPORT_SYMBOL(kfree);
```

二、kmalloc()/kfree()案例实战分析