第 0019 讲 13 创建内存池案例实战分析

一、内存池基础知识

1、内存池是在真正使用内存之前，先申请分配一定数量的、大小相等内存块留作备用。当有新的内存需求时，就从内存池中分出一部分内存块，若内存块不够再继续申请新的内存。

2、优势：内存分配效率得到提升。特性：尽量避免了内存碎片。

3、内存池应用场景

- 高性能服务器领域
- 嵌入式系统领域
- 应用程序频繁分配和释放小块内存

4、创建内存池使用常用

kmem_cache_create 创建内存缓存

kmem_cache_destroy 释放内存缓存

mm > slab_common.c > slab_caches_to_rcu_destroy_workfn(work_struct *)

```
564 struct kmem_cache *
565 kmem_cache_create(const char *name, unsigned int size, unsigned int align,
566         slab_flags_t flags, void (*ctor)(void *))
567 {
568     return kmem_cache_create_usercopy(name, size, align, flags, 0, 0,
569                       ctor);
570 }
571 EXPORT_SYMBOL(kmem_cache_create);
572
```

mempool_create 创建内存池

mempool_destroy 释放内存池

```
mm > C mempool.c > mempool_create(int, mempool_alloc_t *, mempool_free_t *, void *)
258 mempool_t *mempool_create(int min_nr, mempool_alloc_t *alloc_fn,
259                           mempool_free_t *free_fn, void *pool_data)
260 {
261     return mempool_create_node(min_nr,alloc_fn,free_fn, pool_data,
262                                GFP_KERNEL, NUMA_NO_NODE);
263 }
264 EXPORT_SYMBOL(mempool_create);
265
```

mempool_alloc 内存池中分配内存对象

mempool_free 释放分配的内存对象

```
mm > C mempool.c > ...
383
384 void *mempool_alloc(mempool_t *pool, gfp_t gfp_mask)
385 {
386     void *element;
387     unsigned long flags;
388     wait_queue_entry_t wait;
389     gfp_t gfp_temp;
390
391     VM_WARN_ON_ONCE(gfp_mask & __GFP_ZERO);
392     might_sleep_if(gfp_mask & __GFP_DIRECT_RECLAIM);
393
394     gfp_mask |= __GFP_NOMEMALLOC;   /* don't allocate emergency reserves */
395     gfp_mask |= __GFP_NORETRY;  /* don't loop in __alloc_pages */
396     gfp_mask |= __GFP_NOWARN;   /* failures are OK */
397
398     gfp_temp = gfp_mask & ~(__GFP_DIRECT_RECLAIM|__GFP_IO);
```

二、实战案例分析

步骤：创建内存缓存-->创建内存池-->创建内存对象。