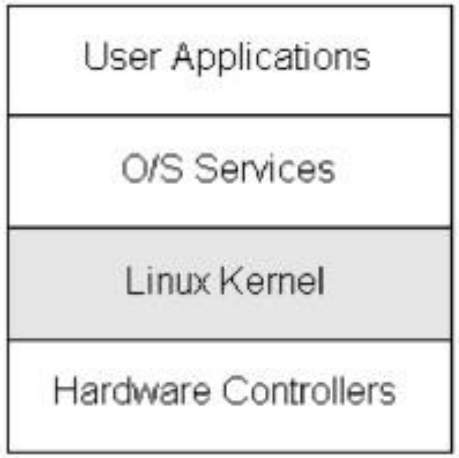
# 第0019讲 1Linux内核内存池案例分析





零声学院讲师：Vico老师

# 学习内容



一、内存池原理

一、内存池数据结构

二、内存池设计与实现

# 一、内存池原理



在实际开发中，为了避免频繁执行malloc/free产生的内存碎片，通常会在程序中设计单独的内存管理模块，即内存池。内存池原理：程序启动时为内存池申请一块比较大的内存区，程序在使用内存时全部都是由内存池进行分配的，不再使用内存交给内存池回收用于再次分配。

内存池常用接口：初始操作、分配内存、释放操作、销毁操作。

# 二、内存池数据结构



Linux内核内存池的数据结构源码分析如下：

include > linux > C mempool.h > ...

```
10  struct kmem_cache;
11
12  typedef void * (mempool_alloc_t)(gfp_t gfp_mask, void *pool_data);
13  typedef void (mempool_free_t)(void *element, void *pool_data);
14
15  typedef struct mempool_s {
16      spinlock_t lock;
17      int min_nr;     /* nr of elements at *elements */
18      int curr_nr;        /* Current nr of elements at *elements */
19      void **elements;
20
21      void *pool_data;
22      mempool_alloc_t *alloc;
23      mempool_free_t *free;
24      wait_queue_head_t wait;
25  } mempool_t;
```

## 1、内存池创建函数源码分析如下：

mm > C mempool.c > mempool_create(int, mempool_alloc_t *, mempool_free_t *, void *)

```
183  mempool_t *mempool_create_node(int min_nr, mempool_alloc_t *alloc_fn,
184                                 mempool_free_t *free_fn, void *pool_data,
185                                 gfp_t gfp_mask, int node_id)
186  {
187      mempool_t *pool;
188      pool = kzalloc_node(sizeof(*pool), gfp_mask, node_id);
189      if (!pool)
190          return NULL;
191      pool->elements = kmalloc_node(min_nr * sizeof(void *),
192                                    gfp_mask, node_id);
193      if (!pool->elements) {
194          kfree(pool);
195          return NULL;
196      }
197      spin_lock_init(&pool->lock);
```

## 2、内存池的使用源码分析如下：

mm > C mempool.c > ...

```
311
312 void *mempool_alloc(mempool_t *pool, gfp_t gfp_mask)
313 {
314     void *element;
315     unsigned long flags;
316     wait_queue_t wait;
317     gfp_t gfp_temp;
318
319     VM_WARN_ON_ONCE(gfp_mask & __GFP_ZERO);
320     might_sleep_if(gfp_mask & __GFP_DIRECT_RECLAIM);
321
```

## 3、内存对象重新放到内存池的源码分析如下：

mm > C mempool.c > EXPORT_SYMBOL(mempool_alloc)

```
391
392 void mempool_free(void *element, mempool_t *pool)
393 {
394     unsigned long flags;
395
396     if (unlikely(element == NULL))
397         return;
398
399     /*
```

# 三、内存池设计与实现



## 设计与实现源码如下：

pmmde.c - vico - Visual Stud

File Edit Selection View Go Run Terminal Help

EXPLORER

vico
- .cache
- .config
- .gnupg
- .local
- .mozilla
- .pki
- .vscode
- Desktop
  - pmmde.c
- Documents
- Downloads
- Music
- Pictures
- Public
- snap

pmmde.c

Desktop > pmmde.c > MAX_POOL_SIZE

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LOOP 5
#define ALLOC_SIZE 8

#define MAX_POOL_SIZE 1024 * 1024
#define BLOCK_SIZE 64

typedef struct memory_map_talbe
{
    char *p_block;
    int index;
    int used;
} Memory_Map_Table;
```

vico@ubuntu: ~/Desktop

File Edit View Search Terminal Help

```
vico@ubuntu:~$ ls
Desktop  Documents  Downloads  examples.desktop  Music  Pictures  Publi
c  snap  Templates  Videos
vico@ubuntu:~$ cd Desktop
vico@ubuntu:~/Desktop$ ls
pmmde  pmmde.c
vico@ubuntu:~/Desktop$ gcc pmmde.c -o pmmde
vico@ubuntu:~/Desktop$ ls
pmmde  pmmde.c
vico@ubuntu:~/Desktop$ ./pmmde
memory_pool_init: total size: 1024, block cnt: 16, block size: 64
Alloc size: 8, Block: (start: 0, end: 0, cnt: 1)
Malloc success
Block: free: start: 0, end: 0, cnt: 1
Alloc size: 8, Block: (start: 0, end: 0, cnt: 1)
Malloc success
Block: free: start: 0, end: 0, cnt: 1
Alloc size: 256, Block: (start: 0, end: 3, cnt: 4)
Malloc success
Alloc size: 512, Block: (start: 4, end: 11, cnt: 8)
Malloc success
Block: free: start: 0, end: 3, cnt: 4
Alloc size: 256, Block: (start: 0, end: 3, cnt: 4)
Malloc success
vico@ubuntu:~/Desktop$
```


办学宗旨：一切只为渴望更优秀的你

办学愿景：让技术简单易懂