

# 广东中兴新支点

## CGEL 内核优化说明

嵌入式操作平台

---

# 目 录

<b>1. 概述.....</b>	<b>1</b>
1.1. 优化特点.....	1
1.2. 优化项.....	1
<b>2. CGEL 内核改造.....</b>	<b>2</b>
2.1. 全局优先级调度.....	2
2.1.1. 修改原因 .....	2
2.1.2. 修改原理及方法 .....	2
2.1.3. 使用方法 .....	4
2.1.4. 对标准内核影响 .....	4
2.2. 实时工作队列.....	5
2.2.1. 修改原因 .....	5
2.2.2. 修改原理及方法 .....	6
2.2.3. 使用方法 .....	7
2.2.4. 对标准内核影响 .....	7
2.3. 内核信号量优先级继承.....	7
2.3.1. 修改原因 .....	7
2.3.2. 修改原理及方法 .....	8
2.3.3. 使用方法 .....	11
2.3.4. 对标准内核影响 .....	11
2.4. ARM NOMMU 文件加载.....	11
2.4.1. 修改原因 .....	11
2.4.2. 修改原理及方法 .....	12
2.4.3. 使用方法 .....	13
2.4.4. 对标准内核影响 .....	13
2.5. IP 转发条件下，邻居表溢出.....	13
2.5.1. 修改原因 .....	13

---

2.5.2.	修改原理及方法 .....	14
2.5.3.	使用方法 .....	15
2.5.4.	对标准内核影响 .....	15
<b>2.6.</b>	<b>idle 任务的 cpu 占用率统计 .....</b>	<b>15</b>
2.6.1.	修改原因 .....	15
2.6.2.	修改原理及方法 .....	15
2.6.3.	使用方法 .....	16
2.6.4.	对标准内核影响 .....	16
<b>2.7.</b>	<b>elf 可执行文件二次加载共享代码段.....</b>	<b>16</b>
2.7.1.	修改原因 .....	16
2.7.2.	修改原理及方法 .....	16
2.7.3.	使用方法 .....	16
2.7.4.	对标准内核影响 .....	17
<b>2.8.</b>	<b>用户态内存申请的直接映射 .....</b>	<b>17</b>
2.8.1.	修改原因 .....	17
2.8.2.	修改原理和方法 .....	18
2.8.3.	使用方法 .....	18
2.8.4.	对标准内核影响 .....	18
<b>2.9.</b>	<b>文件系统大小动态改造.....</b>	<b>18</b>
2.9.1.	修改原因 .....	18
2.9.2.	修改原理及方法 .....	19
2.9.3.	使用方法 .....	20
2.9.4.	对标准内核影响 .....	20
<b>2.10.</b>	<b>精确统计线程非运行时间.....</b>	<b>20</b>
2.10.1.	修改原因 .....	20
2.10.2.	修改原理和方法 .....	20
2.10.3.	使用方法 .....	21
2.10.4.	对标准内核影响 .....	21
<b>2.11.</b>	<b>内存信息统计指标计算.....</b>	<b>21</b>
2.11.1.	修改原因 .....	21
2.11.2.	修改原理和方法 .....	21
2.11.3.	使用方法 .....	27
2.11.4.	对标准内核影响 .....	27

---

<b>2.12. 限制页缓存避免内存耗尽.....</b>	<b>27</b>
2.12.1. 修改原因 .....	27
2.12.2. 修改原理和方法 .....	28
2.12.3. 使用方法 .....	33
2.12.4. 对标准内核影响 .....	33
<b>2.13. Linux 页面分配及回收.....</b>	<b>33</b>
2.13.1. 修改原因 .....	33
2.13.2. 修改原理和方法 .....	38
2.13.3. 使用方法 .....	45
2.13.4. 对标准内核影响 .....	45
<b>2.14. 程序代码段、数据段巨页映射 .....</b>	<b>46</b>
2.14.1. 修改原因 .....	46
2.14.2. 修改原理和方法 .....	47
2.14.3. 使用方法 .....	56
2.14.4. 对标准内核影响 .....	89
<b>2.15. x86-64 用户态地址转物理地址接口 .....</b>	<b>90</b>
2.15.1. 修改原因 .....	90
2.15.2. 修改原理和方法 .....	90
2.15.3. 使用方法 .....	90
2.15.4. 对标准内核的影响 .....	91
<b>2.16. PROC 文件获取 mips 系列 ra 寄存器的显示.....</b>	<b>91</b>
2.16.1. 修改原因 .....	91
2.16.2. 修改原理和方法 .....	92
2.16.3. 使用方法 .....	92
2.16.4. 对标准内核的影响 .....	93
<b>3. 开源 LICENSE 说明 .....</b>	<b>93</b>
3.1. GPL 简介.....	93
3.2. 开发指导.....	93
3.3. 源码获取方式.....	94

# 1. 概述

基于 Linux2.6 内核的版本开发，已成为当今 Linux 开发的主流。广东中兴新支点技术有限公司研发的 CGEL3.0\4.0 (Carrier Grade Embeded Linux: 电信级嵌入式 Linux) 在此基础上对内核进行了完美优化。不仅仅修复了标准内核本身存在的问题，还为满足不同客户需求对内核进行了量身定制，已成为业界领跑者。

## 1.1. 优化特点

为慎重起见，对内核新增的功能均采用配置宏的方式实现，默认为缺省配置，仅当有特殊需求时，才会开启相应的配置宏，从而保证了为不同产品线而特别添加的内核功能，不会对其他产品线的代码产生影响。

## 1.2. 优化项

### ■ CGEL 对标准内核中存在问题进行了如下功能项的修复：

- IP 转发条件下，邻居表溢出
- 文件系统大小动态改造
- 限制页缓存避免内存耗尽

### ■ CGEL 对标准内核的优化主要体现在以下功能项：

- 全局优先级调度
- 实时优先级队列
- idle 任务的 CPU 占用率统计
- elf 可执行文件二次加载共享代码段
- 用户态内存申请的直接映射
- 异常处理框架
- 精确统计线程非运行时间
- 内存信息统计指标计算

## ■ CGEL 移植高版本标准内核或开源补丁：

- 内核信号量优先级继承
- ARMNOMMU 文件加载
- Linux 页面分配及回收

## 2. CGEL 内核改造

### 2.1. 全局优先级调度

#### 2.1.1. 修改原因

##### ■ 问题描述：

在 SMP 环境下，标准 Linux 内核对线程的调度并非完全按照优先级进行，严重影响了内核的实时性。

##### ■ 问题分析：

在 SMP 系统中，每个 CPU 拥有自己单独的运行队列，CPU 在进行调度时只从自己的运行队列中选取任务，因此这就会造成低优先级任务提前获得调度，而高优先级任务迟迟得不到运行的问题。

#### 2.1.2. 修改原理及方法

要解决这个问题，就必须考虑从**任务的唤醒**和**调度机制**上入手。在 SMP 上，若在任务调度时发现其他 CPU 的运行队列中有多个实时任务，那么采用**拉**的方式将正在等待运行的全局优先级最高的任务拉到本地 CPU 上运行，从而确保在每次任务调度时选择的都是全局优先级最高的实时任务。

##### ■ 任务唤醒

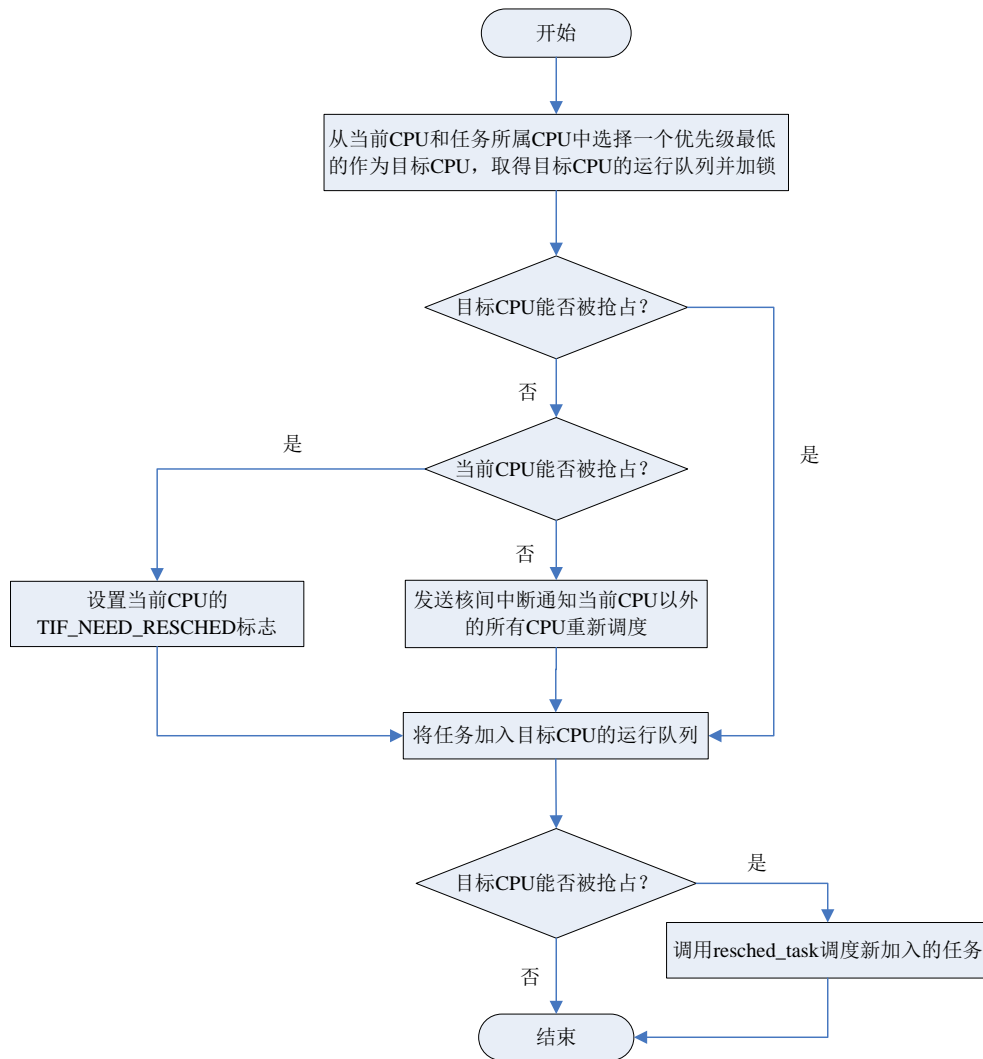


图 2-1 任务唤醒流程

## ■ 任务调度

设置全局变量 `rt_overload`，当某个 CPU 上的实时任务达到 2 个的时候将 `rt_overload` 值加 1，当 CPU 的实时任务等于 1 的时候将 `rt_overload` 值减 1。这个变量的作用即是保存实时任务大于 2 的运行队列数量。

在 CPU 进行调度时，判断若 `rt_overload` 大于 1，那么就表示可能需要通过拉的操作来实现实时任务的平衡。拉操作分如下几步进行：

1. 首先锁住当前 CPU，从当前 CPU 的运行队列中选取优先级最高的任务；
2. 选取一个正在运行的 CPU，如果 CPU 的实时任务数不大于 1 则继续下一个 CPU；

3. 锁住所选 CPU 的运行队列，从运行队列中选取优先级最高的实时任务，但必须遵守以下条件
  - 选择的实时任务不能是正在运行的任务，且该任务允许在当前 CPU 上运行；
  - 若没有实时任务（选择的最小优先级大于 MAX\_RT\_PRIO），则返回 NULL；
  - 若所选任务的优先级高于此 CPU 上当前正在运行的任务，那么可以认为这个任务在目标 CPU 上即将得到运行，则返回 NULL；
4. 比较步骤 1 和步骤 3 选出的实时任务，如果步骤 3 选出的实时任务优先级更高，那么将该任务拉到当前 CPU 的运行队列中，且每拉一次便更新本地 CPU 运行队列的最高优先级；
5. 继续遍历其余的 CPU。

## ■ 实时任务增加的调度时机

采用拉的方式进行实时任务的全局优先级调度，因此需要增加系统的调度点，这样才能保证高优先级的任务尽快得到运行。对于实时任务有如下新增的调度点：

1. 唤醒时如果目标 CPU 不能被抢，而当前 CPU 能被抢，那么设置当前 CPU 的 TIF\_NEED\_RESCHED 标志；
2. 如果目标 CPU 和当前 CPU 都不能被抢，那么发送核间中断通知所有 CPU 重新调度；
3. 在完成任务切换时，如果被切换下来的任务和被切换上去的任务同时都是实时任务，且被切换下来的任务还处于可运行状态，那么发送核间中断通知所有 CPU 重新调度。

### 2.1.3. 使用方法

在 menuconfig 中打开 “**Processor type and features ---> Symmetric multi-processing support**”、**“Processor type and features ---> Multi-core scheduler support”**（此选项默认选中）及 **“Processor type and features ---> RealTime Scheduler support”** 三个选项。

### 2.1.4. 对标准内核影响

相对于标准内核仅能在单 CPU 上进行完全的优先级调度，采用了“全局优先级调度”技术则可以实现多个实时任务在多 CPU 间完全按照优先级进行调度，使得 Linux 在调度机制上更符合实时系统的要求。但同时，全局优先级调度会增加 CPU 的负载，增加的比例大约在 1~2%。

由于所有的代码都通过 CONFIG\_REAL\_PREEMPT 这个宏进行控制，所以只有在 menuconfig 中配置了本补丁的功能时，才会对任务的调度机制产生影响，否则仍然保持标准 Linux 的任务调度流程。



## 2.2. 实时工作队列

### 2.2.1. 修改原因

#### ■ 问题描述：

任务中出现死循环后，串口无响应。

#### ■ 问题分析：

Shell 通过串口响应用户交互，即是能从串口这个终端上读取用户输入的字符。在 Linux 内核中串口是作为一个 tty 终端来处理。tty 的体系结构如下：

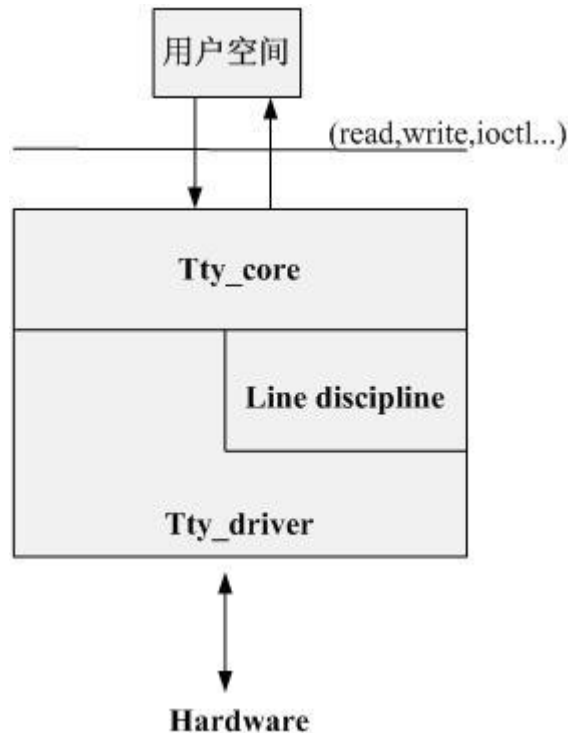


图 2-2 tty 体系结构

一个 IO 设备若要加入这个体系，则需将自己的设备操作添加到一个 `tty_driver` 结构中，作为与 tty 核心层、line discipline（终端线规程）打交道的接口。而用户空间主要是通过设备文件同 `tty_core` 交互，再由 `tty_core` 根据用户空间操作的类型选择是与 line discipline 还是 `tty_driver` 交互。例如设置硬件的 `ioctl` 指令直接交给 `tty_driver` 处理，而 `read` 和 `write` 操作就会交给 line discipline 处理。

可见 Shell 对串口进行系统调用的过程为：

read -> tty\_read -> line discipline 的 read 接口->读取串口驱动 tty\_driver 数据

显然，在用户没有输入时，无数据可读，read 就会一直处于阻塞状态，即是将 line discipline 中的 read 接口函数 read\_chan 放入等待队列 tty->read\_wait，然后触发调度，切换到其它线程去执行了：

```
add_wait_queue(&tty->read_wait, &wait)
set_current_state(TASK_INTERRUPTIBLE)
schedule_timeout(timeout)
```

一旦用户进行终端设备的输入，就会首先在 line discipline 中进行预处理。即是通过调用 receive\_buf() 的回调接口，将输入信息写入到 line discipline 中的一个输入缓存区，从而唤醒 read 阻塞的线程：

```
wake_up_interruptible(&tty->read_wait)
```

但由于调用 receive\_buf() 回调接口的 flush\_to\_ldisc 函数在 tty 初始化时，预先被初始化为一个工作队列结构的入口函数：

```
initialize_tty_struct:
    INIT_DELAYED_WORK(&tty->buf.work, flush_to_ldisc);
```

且此工作队列又是在 tty\_flip\_buffer\_push 函数中启动：

```
schedule_delayed_work(&tty->buf.work, 1);
```

在系统中预定义了两个工作队列：keventd\_wq 和 kblockd，其分别对应的工作线程为 Keventd 和 Kblockd。由于这两个线程的入口函数均为 worker\_thread，所以都是普通的内核线程，在系统中属于非实时任务。而 tty\_flip\_buffer\_push 函数就是在这个非实时任务的上下文中执行的。

通过以上分析可以看出：正是由于工作队列的非实时性，导致当系统中有实时任务处于死循环时，工作队列线程得不到调度运行，进而造成 Shell 任务得不到唤醒，此时系统表现为假死状态。

### 2.2.2. 修改原理及方法

只要提高工作队列所在线程的优先级就可解决串口无响应的问题。提高其优先级的方法如下：

1. 新建一个工作队列，同时提高新建的工作队列的优先级，并且将提高优先级的代码放在 create\_workqueue\_thread 中，而不是 worker\_thread 中；
2. 将与串口相关的工作任务添加到单独的工作队列中，修改代码如下：

```
schedule_delayed_work(&tty->buf.work, 1)
改成：
```

queue\_delayed\_work

- 如此修改的好处是：影响面小，仅提高了串口终端的优先级。

实时工作队列补丁的 patch 文件是：



CGEL 实时工作队列补丁适用于：

- 系统中很多重要服务依赖于工作队列，而工作队列是在非实时线程中运行，一旦系统中有实时任务占用过多 CPU，导致这些工作队列得不到执行时，则需提供实时工作队列；
- 目前支持将 TTY 和键盘相关的服务加入到实时工作队列中，以解决控制台得不到响应的问题；
- 允许在配置时，分别设置实时工作队列所在线程的优先级。

### 2.2.3. 使用方法

1. 在 menuconfig 中打开 “**Enable realtime workqueue**” 选项。
2. 若系统使用串口作为控制台，请再打开 “**enable tty realtime workqueue**” 子选项。
3. 若系统使用键盘作为控制台，请再打开 “**enable keyboard realtime workqueue**” 子选项。
4. 可根据需要设置实时工作队列线程的优先级，默认是 95，最高为 99。

### 2.2.4. 对标准内核影响

由于所有的代码都通过 **CONFIG\_RT\_WQ**、**CONFIG\_RT\_WQ\_TTY**、**CONFIG\_RT\_WQ\_KBD** 这三个宏进行控制，所以只有在 menuconfig 中配置了本补丁的功能时，才仅会对键盘、串口工作队列的优先级产生影响，否则在默认情况下，这三个宏都未生效，系统走的仍然是标准内核的处理流程。

## 2.3. 内核信号量优先级继承

### 2.3.1. 修改原因

#### ■ 问题描述：

在实时操作系统中，优先级反转问题可能造成高优先级任务的响应变慢，进而影响整个系统的实时

性能。

## ■ 问题分析

在标准 Linux 内核中，由于信号量不支持优先级继承，因此可能存在优先级反转问题。

以下演示一个发生该问题的场景：

假设任务 A、B、C，优先级从低到高顺序为  $A < B < C$ ：

1. 首先，任务 A 通过系统调用进入内核，并获取了某信号量；此时任务 C 优先级最高，抢占了 A；
2. 由于任务 C 在运行过程中也需要获取该信号量，但此时信号量已被 A 持有，所以任务 C 发生了阻塞；
3. 任务 A 获得信号量，在等待信号量期间，由于发生了抢占调度，使得具有中间优先级的任务 B 运行。

此时，就发生了优先级反转，即中优先级的任务 B 占用 CPU，而高优先级的任务 C 处于等待信号量的状态。若任务 B 又长时间占有 CPU，则会导致任务 A 一直得不到调度无法释放信号量，进而使得高优先级任务 C 也无法运行。

### 2.3.2. 修改原理及方法

在由 Linux 核心开发成员 Ingo Molnar 提出的 Linux 实时补丁中已提供了优先级反转问题的解决方案，以提高系统实时性能。具体内容包括：spinlock 可抢占，信号量支持优先级继承，中断线程化等。目前，广东中兴新支点技术有限公司也已经在 CGEL 中完成对信号量优先级反转相关补丁的移植。

该补丁是通过使用 `rt_mutex` 互斥量的方式来实现信号量。`rt_mutex` 可以看成是一种互斥锁，但具有更为强大的功能，其特点如下：

- `rt_mutex` 锁支持优先级继承，实现了优先级继承算法，能有效解决优先级反转问题；
- 若高优先级任务阻塞在低优先级任务的互斥锁上，优先级继承算法会提升持有锁的低优先级任务的优先级，使得低优先级任务能够尽快释放锁，让高优先级任务更快得到运行；
- 一个 `rt_mutex` 锁只有唯一的持有者，但是一个任务可以同时持有多个 `rt_mutex` 锁；
- `rt_mutex` 锁维护一个按照优先级排序的等待队列，在锁释放时，会将等待队列中具有最高优先级的任务唤醒；
- `rt_mutex` 支持死锁检测。

下面通过一个简单的例子，具体描述 `rt_mutex` 是如何解决优先级反转问题：

有五个任务 A、B、C、D、E，以及 4 个互斥锁 L1、L2、L3、L4，假设：

```

A 持有: L1
  B 阻塞在 L1
    B 持有 L2
      C 阻塞在 L2
        C 持有 L3
          D 阻塞在 L3
            D 持有 L4
              E 阻塞在 L4

```

整个依赖关系链如下：

**E->L4->D->L3->C->L2->B->L1->A**

由于一个任务只能阻塞在一个互斥锁上，但一个任务又同时可以拥有多个互斥锁，因此整个依赖链只可能汇聚而不可能分支。

为了示例两个依赖链如何汇聚，假设还有任务 F 和锁 L5，B 拥有 L5，且任务 F 阻塞在 L5 上：

**F->L5->B->L1->A**

如下为两个依赖链合并后的链表：

**E->L4->D->L3->C->L2-> B->L1->A**

|

**F->L5**

rt\_mutex 优先级继承算法即是基于上面的依赖链关系。假设任务 E 需要获取锁 L4，由于 L4 已经被 D 持有，因此 E 将发生阻塞。此时优先级继承算法将沿着该依赖链，从左至右进行遍历，调整整个依赖链上任务的优先级。通过这样的方法，使得持有锁的低优先级任务能够得到运行并尽快释放锁，能有效解决优先级反转问题。

另外，在每个锁上维护了一个按优先级排队的等待队列，在释放锁以后，会唤醒队列头上优先级最高的任务。

具体实施方案是：

1. 在 menuconfig 中增加一个配置项：用户可选择是否配置信号量支持优先级继承。所有的修改使用配置宏 CONFIG\_SEMAPHORE\_PI 进行控制；
2. 定义 semaphore 信号量结构，采用 rt\_mutex 实现；
3. 定义该类型信号量操作的接口函数；


4. 对原来的信号量重新命名，在其名字前添加 “**compat\_**” 前缀；
5. 对原来信号量操作函数加上 “**compat\_**” 前缀。
6. 统一 semaphore 和 compat\_semaphore 的操作接口。

例如将 down() 定义为如下的宏：

```
#define down(sem) \  
    PICK_FUNC_1ARG(struct compat_semaphore, struct semaphore, \  
        compat_down, rt_down, sem)
```

该宏可以根据信号量类型不同调用不同的函数接口，如果是 compat\_semaphore 类型的信号量调用 compat\_down()，如果是 semaphore 类型信号量则调用 rt\_down() 接口。

如此定义的好处：不用大量修改已有的操作信号量的代码，因为信号量的操作接口仍然不变。

 提示：如果用户需要使用以前的信号量，此时需要将信号量定义为 compat\_semaphore 类型。

## ■ 新旧信号适用场景

新旧信号量存在以下区别，

- 新的信号量支持优先级反转，旧信号量不支持；
- 新的信号量只能用于互斥，而旧的信号量还可用于同步；
- 新的信号量比旧的信号量复杂，在信号量竞争不激烈的时候，性能损失不大。但若应用比较复杂，且容易产生竞争条件的话，可以考虑使用旧的信号量。

针对以上区别，新旧信号有不同的使用场合：

**旧信号适用于：**

1. 使用信号量进行同步。典型的同步情形是：首先初始化一个空的信号量，然后某个任务调用该信号的 down 接口，阻塞在该信号量上，等待中断或者其他任务调用 up 接口将阻塞在该信号量上的任务唤醒。这种情况下必须使用旧的信号量。

补丁中个别地方使用 compat\_semaphore 类型就是这个原因。

2. 对性能要求高，容易产生信号量竞争，但是对优先级反转不是很敏感的情况。

**新信号适用于：**

除以上两种情况外，其他情况下都建议使用新的信号量。

### 2.3.3. 使用方法

在 menuconfig 中打开 “Semaphore support priority inherit ---> Semaphore support priority inherit ” 选项。

### 2.3.4. 对标准内核影响

所有的代码都通过 **CONFIG\_SEMAPHORE\_PI** 进行控制，所以仅当在 menuconfig 中配置了本补丁的功能时，才会影响信号量操作接口，间接影响到系统的调度行为。否则，在默认情况下，这个宏并未生效，系统走的仍然是标准内核的处理流程。



提示：由于本补丁是成熟的开源社区补丁，已经过了长期测试，因此建议在不影响系统性能，且对实时响应要求较高的情况下，可以打开本功能。



提示：当配置内核信号量优先级继承时，在多进程/多线程竞争用户态 PI 锁时，需要用户确保锁的竞争频率不要太高。这是因为为了提高系统的响应度，尤其是高优先级任务的实时性能，信号量的处理逻辑变的更为复杂，增加了对应的系统开销，对整个系统的吞吐量略有影响。

## 2.4. ARM NOMMU 文件加载

### 2.4.1. 修改原因

#### ■ 问题描述

在 ARM NOMMU 体系结构中进行文件加载时，发现有大量的内存被浪费。

#### ■ 问题分析

在 ARM NOMMU 体系结构中，内核在加载 bflt 格式的文件时，会为代码段、数据段、BSS 段、堆栈段统一分配一块内存区域。在系统加载 bflt 文件时候，会通过一系列的内核调用 `do_mmap->do_mmap_pgoff->do_mmap_private->kmalloc` 来分配内存。而 kmalloc 采用的 slab 内存管理算法，其缺省

返回的内存大小均为 2 的 n 次方。即是说，通过 slab 申请的内存，将有 2 的 (n-1) 次方被浪费。

通过分析内核源代码发现，内核会将多分配到的这部分内存归结到堆中。表明上看来，当应用程序调用 malloc 函数申请内存时，malloc 函数返回的指针会指向进程的堆区域，因而这些多余的内存会被利用起来。但通过测试后发现，在应用程序中，无论调用 malloc 分配多大的内存大小，返回的指针都不会指向该进程的堆区域，也就意味着这部分内存被浪费了。

如何才能将堆区域中的这部分剩余内存利用起来，通过分析与实现 malloc 有关的工具链提供的库 uclibc 的源代码后发现，只有在定义了宏 **MALLOC\_USE\_SBRK** 后，才会在堆中进行相应的内存分配。而宏 **MALLOC\_USE\_SBRK** 依赖于系统是否带有 MMU 控制器，若无此控制器，该宏不会打开。当 **MALLOC\_USE\_SBRK** 不被打开时，malloc 函数是通过 **mmap** 系统调用来动态分配内存给应用程序，因而不会利用应用程序堆中的内存。

### 2.4.2. 修改原理及方法

开源社区提供的“Unbreak no-mmummap patch”补丁已解决此问题。其基本思想是：将 kmalloc 多分配的内存，通过 **free\_pages** 归还给系统。但该补丁会同时引发了内存碎片的问题。关于该问题的详细说明，请参见 <http://kerneltrap.org/mailarchive/linux-kernel/2007/6/22/107735>。

广东中兴新支点技术有限公司通讯 OS 平台在此基础上，对此问题进行了优化：

1. 将释放页的方法，从 **free\_pages** 改成 **\_\_free\_pages\_ok**。从而保证多分配的内存是被归还给伙伴系统而非内存缓冲池；
2. 并非将所有浪费的内存都归还给系统，而仅选择大块内存。为此，增加 **sysctl\_nr\_trim\_pages** 控制参数。这个参数的默认值是 64，即只归还超过  $64 * 4K = 256K$  的块。低于 256K 的块被浪费。**sysctl\_nr\_trim\_pages** 参数可以通过 **/proc/nr\_trim\_pages** 文件来进行控制。这个参数的取值范围是：  
 **$1 < nr\_trim\_pages < 1024$**
3. 可以控制是否归还浪费的内存。由于一般的命令都是一启动就退出，所以运行命令时，不归还多申请的内存，而是等命令运行结束后自行释放内存块。如果是启动后台命令，可以在运行程序时加上 **-splitpages** 参数，这样，程序启动后，浪费的内存会及时归还给系统。

---

 **提示** 对于 **"/sbin/init" , "/bin/msh" , "/bin/sh" , "/bin/sw"** 等几个程序 , 系统认为是后台程序 , 不加 **-splitpages**

参数也会归还内存。

---



### 2.4.3. 使用方法

在 menuconfig 中打开 “**Kernel Features** --->**Enable NOMMU memory patch**” 选项。

如前所述，除了“/sbin/init”，“/bin/msh”，“/bin/sh”，“/bin/sw”等几个程序外，若想在启动应用程序后，释放多申请的内存，还需要在启动参数中加上 “-splitpages”。

### 2.4.4. 对标准内核影响

本补丁仅对 NOMMU 类型的 CPU 有效，并且通过 **CONFIG\_NOMMU\_MEM\_PATCH** 宏进行控制。该宏默认情况下是打开的，但可通过 menuconfig 将其关闭。

本补丁已经被最新版本的 Linux 官方内核采用，因此，使用风险甚小。

## 2.5. IP 转发条件下，邻居表溢出

### 2.5.1. 修改原因

#### ■ 问题描述

当 Linux 主机同时满足以下条件时，可能出现邻居表溢出故障，从而导致系统网络不正常：

- 主机打开了 IP 转发功能；
- 要转发包的目的地 MAC 地址是多播或者广播地址；
- 要转发包的目的地 IP 地址与主机的 IP 地址在同一个网段；
- 在一定时间内，分别向不同的 IP 地址广播；
- 主机属于 A 类或者 B 类网络，IP 范围超过了 arp 表的大小。

#### ■ 问题分析

经跟踪定位，发现此故障是标准 Linux 的一个缺陷。其产生原因如下：

主机在转发包前，会调用 **ip\_route\_input** 确定路由。**ip\_route\_input** 函数首先调用 **arp\_bind\_neighbour** 向 arp 表中添加了一个 nud\_state 为 0 的条目。一般情况下，当向一个不存在的主机转发包时，nud\_state 会很快变成 NUD\_INCOMPLETE，然后变成 NUD\_FAILED，并很快从邻居表中删除，而不会造成邻居表溢出。

但是，在 **ip\_route\_input\_slow** 函数中，当在邻居表中添加一个条目后，会继续调用处理函数。在转

GDLC 版权所有 不得外传

发包时，最终回调的函数是 **ip\_forward**。ip\_forward 执行了以下操作：

```
int ip_forward(struct sk_buff *skb)
{
    .....
    /**
     * 特殊条件下，此判断条件为真，函数直接退出了。
     */
    if (skb->pkt_type != PACKET_HOST)
        goto drop;
    .....
}
```

分析代码后发现，只要检查到满足“被转发包的 MAC 地址是广播或多播地址”这一个条件时，函数 **ip\_forward** 就会直接退出，不再进行包转发操作。显然此时 **ip\_forward** 并未执行到 **ip\_forward\_finish**，也就没有执行 **ip\_output**。但是 arp 条目的 nud\_state 状态改变是在 **ip\_output** 中进行的。这就意味着，判断出包不可进行转发时，并未及时在邻居表中删除对应的 nud\_state 条目，无用信息的过多堆积，最终导致邻居表溢出。

### 2.5.2. 修改原理及方法

经分析，只要在向邻居表添加条目前，先判断一下被转发包的 MAC 地址就可避免邻居表未被删除的故障发生。通过对文件 **net/ipv4/rout.c** 进行修改实现（如下的蓝色代码）：

```
if (!IN_DEV_FORWARD(in_dev))
    goto e_hostunreach;
if (res.type != RTN_UNICAST)
    goto martian_destination;

/**
 * avoid to overflow of neighbor table
 * author: xiebaoyou172958@zte.com.cn
 */
if ((skb->pkt_type == PACKET_BROADCAST) ||
    (skb->pkt_type == PACKET_MULTICAST))
{
    goto martian_source;
}
```

```
err = ip_mkroute_input(skb, &res, &fl, in_dev, daddr, saddr, tos);
if (err == -ENOBUFS)
    goto e_nobufs;
```

### 2.5.3. 使用方法

本补丁直接打入内核，无需进行配置。

### 2.5.4. 对标准内核影响

由于本补丁是针对标准内核缺陷进行的修改，因此没有通过配置宏进行控制，而是直接对 IP 转发的流程进行修改，但已经过缜密的测试、验证，补丁是安全可靠的。

## 2.6. idle 任务的 cpu 占用率统计

### 2.6.1. 修改原因

#### ■ 问题描述

实际应用中，项目需要 IDLE 任务调度精确计时，但 IDLE 任务做不到。


#### ■ 问题分析

对 IDLE 任务来说，无论是否在编译内核时打开了 CFS 功能选项，都不会在 IDLE 任务调度时进行精确计时，也就导致：

- IDLE 任务的执行时间统计只能精确到 TICK；
- 在极端情况下，IDLE 任务的实际执行时间与统计结果之间误差较大。

### 2.6.2. 修改原理及方法

本补丁参照 CFS 补丁的方法，在调度 IDLE 任务时，记录下 IDLE 任务调入、调出的时间，并在 /proc/stat 中，输出准确的统计时间。

 提示：CFS 补丁已经打入 2.6.24 官方内核，是一个非常成熟稳定的补丁。

### 2.6.3. 使用方法

在 menuconfig 中打开 “**Kernel hacking** ---> **Collect scheduler debugging info**” 选项。

### 2.6.4. 对标准内核影响

仅对 IDLE 任务运行在 proc 文件中的输出结果进行修改, 并且通过 **CONFIG\_SCHEDSTATS** 宏控制。对内核关键流程并无影响。

## 2.7. elf 可执行文件二次加载共享代码段

### 2.7.1. 修改原因

#### ■ 问题描述

应用程序的二次解压, 如何使得内核识别前后两份文件是同一程序的拷贝?

#### ■ 问题分析

在某些项目中, 由于存储介质比较紧张, 一般是将应用程序压缩后存放在存储介质中, 在运行前再将其解压, 待应用程序启动后, 将其删除。

但是, 如果在应用程序运行过程中, 再次解压同一文件, 并启动同一应用程序, 则会出现两份代码段同时存在于内存中的情况。这是因为: 内核会认为两份文件并不相同。为此, 需要调整内核, 使内核识别前后两份文件是同一程序的拷贝, 并直接在内核中共享前一份应用程序的代码段。

### 2.7.2. 修改原理及方法

1. 采用计算应用程序 CRC32 校验和的办法, 让应用程序在第二次启动时, 从内存中找到已有代码段;
2. 应用程序第一次加载时, 将所有代码段都读取到内存中, 并且锁住以避免 Linux 将代码换出。在第二次加载时, 直接复制第一次的页表项, 而不再从文件中读取代码, 以达到代码段共享的目的。

### 2.7.3. 使用方法


在 menuconfig 中打开 “**Allocate Pages at brk/mmap** ---> **Enable allocating pages when process**

---

address mapped---> Enable ELF binary be shared when reloading” 功能选项。

打开本功能选项后，还需要在启动应用程序时，加上“-elfbinaryshare”参数。

---

 提示：一旦应用程序进行了代码段共享，就会在内存中锁住代码，禁止交换到物理介质上（因为此时物理介质上的文件可能已经不存在），因此，OS 提供的换页功能无法应用到该应用程序。

---

#### 2.7.4. 对标准内核影响

所有的代码都通过宏 **CONFIG\_ELF\_BINARY\_SHARE** 进行控制，所以仅当在 menuconfig 中配置了本补丁的功能时，才会修改内核文件启动过程、代码段加载及映射过程。否则，在默认情况下，这个宏并未生效，系统走的仍然是标准内核的处理流程。在存储介质较充裕的项目中，可不开启此功能项。

同时，本功能经过充分的测试，并通过实际项目的检验，因此可根据项目实际情况放心选用。

## 2.8. 用户态内存申请的直接映射

### 2.8.1. 修改原因

#### ■ 问题描述

内存换页功能关闭，导致内存紧张，产生缺页异常，出现 oom-kill 后果。

#### ■ 问题分析

在嵌入式设备中，为确保任务运行的实时性，一般都关闭了内存换页功能。这会造成在内存紧张时，易产生缺页异常并进入内核的 **oom-kill** 流程。在普通桌面系统中，由于 oom-kill 而杀死用户态进程的后果还不是太严重，但是在嵌入式系统中，一般不容许发生 oom-kill。

一般，在系统内存不足时，应用程序调用 malloc 分配内存。但内核并不会直接为应用程序分配物理内存，而是为其分配一段虚拟地址空间，直到应用程序实际访问分配的虚拟内存时，才会为其分配实际的物理内存。这种方式和 VxWorks 有较大差异，造成的直接后果是：应用程序通过 malloc 分配内存，看起来是成功的，实际访问时，却可能产生 oom-kill 这样的后果。

### 2.8.2. 修改原理和方法

1. 修改内核的 **brk/mmap** 系统调用。在应用程序调用 **malloc** 分配虚拟地址空间时，同时为其分配物理内存，建立好虚拟内存与物理内存之间的映射。若实际物理内存不足，则直接向应用返回 **NULL**；
2. 为防止分配的页面被换出到物理介质，还将分配的物理页面锁在内存中；
3. 为避免在内存不足时，发生 oom-kill，进而将进程杀死，对 oom-kill 流程进行修改：当出现内存不足时，不是杀死进程，而是根据上下文，向调用者返回 **NULL**，或者等待其他进程释放内存。

### 2.8.3. 使用方法

在 menuconfig 中打开 “**Allocate Pages at brk/mmap ---> Enable allocating pages when process address mapped**” 功能选项。

### 2.8.4. 对标准内核影响

所有代码都通过宏 **CONFIG\_MAPPING\_ALLOC** 进行控制。当不配置该功能时，对标准内核没有任何影响。

本补丁经过充分测试，通过实际项目检验。

## 2.9. 文件系统大小动态改造

### 2.9.1. 修改原因

#### ■ 问题描述

由于内存文件系统 **ramfs** 没有限制文件系统的大小，所以当以 **ramfs** 作为系统的根文件系统时，一旦应用向该文件系统写入过多文件时，很容易导致文件系统耗尽系统内存，从而危及整个系统的正常运行。

#### ■ 问题分析

不仅仅要防范向根文件系统写入过多内容，同时还需要找寻方法来避免内存文件系统对内存资源的无限制耗用。目前，内核提供三种以内存最为最终存储设备的根文件系统机制，三种机制可应用于不同的场景，拥有各自的优缺点：

表 2-1 根文件系统实现机制对比

➤ 根文件系统 机制	➤ 优点	➤ 缺点
➤ ramdisk	➤ 文件系统大小固定，可避免根文件系统内容过多	➤ 制作过程相对比较繁琐，且需要一个文件系统进行格式化及 mount； ➤ 占用固定大小内存导致“内存磁盘”内未使用的部分也占用系统内存，带来浪费； ➤ 存在页缓存，存在不必要的内存拷贝； ➤ 对于不同单板、应用，ramdisk 大小难以预先计算，因此制作合适大小的内存磁盘。
➤ tmpfs	➤ 文件系统的大小可动态变化 ➤ 速度块 ➤ 可以限制文件系统的大小 ➤ 文件系统空间大小可动态修改 ➤ 这个根文件系统的切换应用是不可见的	➤ 需要人为确定文件系统空间上限 ➤ 需要对内核稍作修改
➤ ramfs	➤ 具有 tmpfs 描述的所有优点	➤ 对内核修改量大，并且为非标准方式

由以上分析可知，采用 **tmpfs** 作为根文件系统是最好的解决方案。

### 2.9.2. 修改原理及方法

由于采用 tmpfs 的方式需要人为确定文件系统空间的上限，但这本身就是无法避免的。我们只需提供操作系统为应用设置上限值的一种手段，且按这个值限制文件系统空间的扩展便可以了。

同时，我们可以很方便的通过使用 mount 的 remount 选项调整文件系统空间上限（当然也可以通过编程方式调用 mount()，传入正确的参数轻易实现）。当设置的上限小于文件系统已占用的空间时，mount 将失败。


例如：

```
mount -o remount,size=32m #调整根文件系统空间上限到 32m，不对文件系统已有内容产生影响
```

### 2.9.3. 使用方法

1. 版本构建时，将文件系统按现有的 cpio 等方式构建成 initrd，并打入 version.bin 中；
2. 依据系统目前根文件系统的大小及后续允许写入根文件系统的文件大小，通过 **remount** 调整根文件系统可占用内存空间的上限值。根文件系统大小等信息可调用 **statfs("/", &buf)** 获得。

---

 **提示：**管理进程可在系统运行过程中继续调整。

---

对文件系统的动态调整也可采用如下方式，但实现前需考虑应用在其他内核的兼容性问题：

- 由于内核将相关信息导入到 /proc 文件系统中，所以应用可修改对应的 /proc 文件来实现对上限值的修改；
- 目前以内存大小来设置上限值，也可修改为应用占总内存的百分比来设置这个上限值。

### 2.9.4. 对标准内核影响

对内核的修改不多，且均利用内核现有机制实现。

## 2.10. 精确统计线程非运行时间

### 2.10.1. 修改原因

#### ■ 问题描述

在任务运行时，需要精确统计某个函数或某段代码的运行时间，避免执行中间有任务调度发生，而导致统计的运行时间不准确。

#### ■ 问题分析

需要获取任务的精确运行时间，则可以反向通过精确统计任务处于非运行状态的时间而得到。

### 2.10.2. 修改原理和方法

为每一个线程在 **proc** 中添加一个文件，用于放置线程累积的非运行时间信息。通过在任务控制块中添加字段，记录线程的累计非运行时间。通过调用 **schedule** 函数内中新增的统计空闲时间的函数



taskoutcpu\_statistics, 当线程被调出时调用 taskoutcpu\_depart(prev), 记录下此刻时间点, 当再次被调入运行时调用 taskoutcpu\_arrive(next), 记录下此刻时间点, 则可统计出本次非运行的时间间隔, 累加到该线程的非运行时间统计中, 供 proc 查询。

```
if (prev != rq->idle)
    taskoutcpu_depart(prev);
if (next != rq->idle)
    taskoutcpu_arrive(next);
```

对应每个线程的非运行时间不仅仅是一次时间统计, 而是任务运行过程中一段时间内得到的一个非运行时间的累积统计值, 因此这个值是不断累加的。

### 2.10.3. 使用方法

在 menuconfig 中打开 “**Show the time task out off cpu**” 功能选项。

### 2.10.4. 对标准内核影响

所有代码都通过宏 **CONFIG\_TASKOUTCPU\_STATISTICS** 进行控制。当不配置该功能时, 对标准内核没有任何影响。

## 2.11. 内存信息统计指标计算

### 2.11.1. 修改原因

#### ■ 问题描述

在标准 Linux 内核中, 未对内存的各项信息提供一个完整的统计显示。

#### ■ 问题分析

为方便用户对内存信息有一个整体的驾驭, 在 CGEL 内核中按照不同的分类分别对内存信息提供了信息统计功能。

### 2.11.2. 修改原理和方法

在 proc 文件系统中新增以下六类内存信息统计功能:

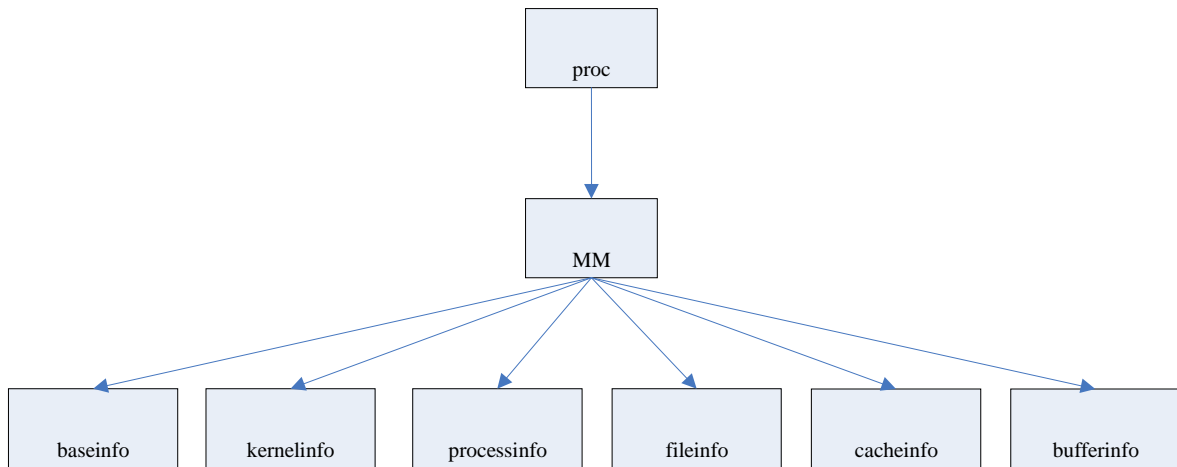


图 2-3 内存信息统计

■ **baseinfo:** 显示内存当前基本信息

```
# cat /proc/mm/baseinfo
Memory:      2048000 kB
Available:   2027944 kB
Free:        1983664 kB
HighMem:     1130432 kB
LowMem:      897512 kB
UserSpace:   39739 kB
KernelSpace: 4541 kB
PageCache:   2216 kB
```

图 2-4 baseinfo 信息统计


1. 物理内存大小: num\_physpages;
2. 可用物理内存大小: totalram\_pages;
3. 剩余物理内存: global\_page\_state(NR\_FREE\_PAGES);
4. 高端内存大小: totalhigh\_pages;
5. 低端内存大小: totalram\_pages - totalhigh\_pages;
6. 用户空间占用内存大小;

遍历各进程线性区间，对存在物理内存中的页进行累加。对于 `page_mapcount(page)` 大于 2 的页被认为是多进程共享，累加量为 `PAGE_SIZE / page_mapcount(page)`。对于内核与进程共享的页将被记入用户空

间，但对于进程退出且仍然存在的共享页，此时将计入内核空间。

进程占用物理内存 = 进程私有物理内存页面总和 + 进程共享物理内存页面 1 / 该页面被共享次数 + 进程共享物理内存页面 2 / 该页面被共享次数……

7. 内核空间占用物理内存大小：当前已使用物理内存 - 用户空间占用物理内存大小。

 提示：以上均为物理内存统计。除用户空间统计采用遍历进程累加外，其余均有现成指标可用。由


于内核空间占用大小采用“已用内存 - 用户空间占用”的方法，因此统计有一定误差。

#### ■ kernelinfo：显示内核使用内存基本信息

```
# cat /proc/mm/kernelinfo
SlabTotal:      1968 kB
Vmalloc:       114680 kB
Buffers:        0 kB
Cached:         2216 kB
SwapCached:    0 kB
Reserved:      19604 kB
```

图 2-5 kernelinfo 信息统计

1. slab 占用内存信息：global\_page\_state(NR\_SLAB\_RECLAIMABLE) + global\_page\_state(NR\_SLAB\_UNRECLAIMABLE);
2. vmalloc 使用内存情况：get\_vmalloc\_info 获得 vmalloc 相关分配信息；
3. buffers 占用内存大小：nr\_blockdev\_pages 函数可获得块设备占用物理页数量；
4. cache 占用内存大小：文件映射页 - buffers 占用物理内存 - 交换缓存 swapcached
5. swapcached 占用内存大小：total\_swapcache\_pages；
6. 保留内存大小：遍历所有页框，累加具有保留页框属性的页。


 提示：除 vmalloc 外，其余为物理内存统计。除保留内存大小是以遍历各页框属性得到外，其余均从现成指标获得。

## ■ processinfo: 进程内存使用信息显示

#	cat	/proc/mm/processinfo						
PID	Name	RSS	CODE	DATA	LIB	ANON	STACK	
::	SHARE	IO						
1	init	1824 kB	1412 kB	8 kB	0 kB	412 kB	84 kB	
::	0 kB	0 kB						
696	telnetd	1824 kB	1412 kB	8 kB	0 kB	412 kB	84 kB	
::	0 kB	0 kB						
698	sw	34608 kB	1412 kB	8 kB	0 kB	33196 kB	84 kB	
::	0 kB	0 kB						
699	ash	1824 kB	1412 kB	8 kB	0 kB	412 kB	84 kB	
::	0 kB	0 kB						
704	cat	1824 kB	1412 kB	8 kB	0 kB	412 kB	84 kB	
::	0 kB	0 kB						

图 2-6 processinfo 信息统计

1. 进程占用物理内存大小：进程内存管理结构体中已有记录值相加，`mm->_anon_rss + mm->_file_rss`;
2. 进程代码段占用内存大小：累加属于代码段线性区间的物理页数量；
3. 进程数据段占用内存大小：累加数据数据段线性区间的物理页数量；
4. 进程使用库占用内存大小：进程各可执行线性区间物理页数量 - 进程代码段物理页数量；
5. 进程匿名映射占用内存大小：`mm->_file_rss`;
6. 进程堆栈占用内存大小：累加属于进程堆栈线性区间的物理页数量；
7. 进程共享内存大小：累加属于进程具有共享属性线性区间的物理页数量；
8. 进程 IO 映射占用物理内存大小：累加属于进程 IO 线性区间的物理页数量。


 提示：以上指标均针对物理内存，采用遍历各进程线性区，根据相应属性获得。

## ■ cacheinfo: 页缓存内存占用信息显示（根据所在管理区布局显示）

#	cat	/proc/mm/cacheinfo						
Zone:	PageCaches	SwapCaches	Locked	WriteBack	Dirty	Mapped	Active	
HighMem	:	2216 kB	0 kB	0 kB	0 kB	0 kB	1688 kB	1548 kB

图 2-7 cacheinfo 信息统计

1. 指定管理区页缓存大小：遍历页框，根据 page\_mapping()返回值、管理区属性累加；
2. 页缓存中用于交换缓存的大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否属于交换缓存累加；
3. 被锁定的页缓存大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否被锁累加；
4. 处于回写状态的页缓存大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否回写累加；
5. 脏页大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否脏页累加；
6. 已与用户进程建立映射关系的页缓存大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否映射到用户空间累加；
7. 处于活动状态的页缓存大小：遍历页框，根据 page\_mapping()返回值、管理区属性及是否处于活动状态累加。


 提示：以上指标针对物理内存，采用遍历各页框属性获得。

■ fileinfo: 页缓存内存占用信息显示（以文件为单位显示）

```
# cat /proc/mm/fileinfo
Name          SIZE      COUNT(3)
busybox      1200 kB   10
cat           4 kB     0
inittab       4 kB     0
rcS           4 kB     0
ash           4 kB     0
shm.out      416 kB    2
sw            4 kB     0
sh            4 kB     0
ifconfig      4 kB     0
mount         4 kB     0
telnetd       4 kB     0
fstab         4 kB     0
profile       4 kB     0
.ash_history   4 kB     0
init          4 kB     0
ps            4 kB     0
passwd        4 kB     0
```

图 2-8 fileinfo 统计信息

1. 占有物理内存的文件名（其中 file（unmapping）为已取消映射关系，但仍旧使用内存的一类文件）；
2. 该文件占用内存实际大小：遍历页框，对 page\_mapping()不为空的页，其 mapping（address\_space 结构）字段中的 nrpages 成员记录了此文件实际占用物理内存大小；
3. 文件 dentry 使用计数（dentry 已有现成记录）。

 提示：由于 fileinfo 的信息量可能会比较大，应此使用的 proc 参数/proc/sys/vm/file\_show\_limit 进行控制，初始只显示页缓存占用量大于 10kB 的文件。

#### ■ bufferinfo：块设备缓存占用内存信息显示

1. 块设备名称：通过块设备相关 dentry 结构获得名称；
2. 块设备占用物理内存大小：通过块设备 inode 得到 i\_mapping，其 nrpages 字段可以获得占用内存大小。

以上六类为 CGEL 内核中新增的内存信息统计功能，同时，又针对原有的 zoneinfo 和 show\_mem 两类信息显示进行了改进。

#### ■ zoneinfo

1. 管理区被 kswapd 扫描的最近一次开始时间：在管理区结构体 zone 中增加时间字段，在 kswapd 开始扫描该管理区时记录时间值；
2. 管理区被 kswapd 扫描的最近一次结束时间：在管理区结构体 zone 中增加时间字段，在 kswapd 扫描该管理区结束时记录时间值；
3. 由于管理区内存不足而最近一次唤醒 kswapd 的时间：在管理区结构体 zone 中增加时间字段，在分配函数针对该管理区内存不足情况唤醒 kswapd 时记录时间值；
4. 管理区被 kswapd 扫描的次数：在管理区结构中增加字段，在 kswapd 扫描该管理区结束时累加该值。

#### ■ show\_mem

1. 管理区被 kswapd 扫描的最近一次开始时间；
2. 管理区被 kswapd 扫描的最近一次结束时间；
3. 由于管理区内存不足而最近一次唤醒 kswapd 的时间；
4. 管理区被 kswapd 扫描的次数；
5. 管理区文件映射大小；
6. 管理区匿名映射大小。

### 2.11.3. 使用方法

在 menuconfig 中打开 “**Show mem information in detail**” 功能选项。

### 2.11.4. 对标准内核影响

所有代码都通过宏 **CONFIG\_MM\_INFO** 进行控制。当不配置该功能时，对标准内核没有任何影响。

## 2.12. 限制页缓存避免内存耗尽

### 2.12.1. 修改原因

#### ■ 问题描述

在 ZXR10-T8000 的系统测试过程中，各测试环境 MPUF 单板经常出现内存耗尽的问题，尤其当启动单板后不久便出现内存不足，而这种情况的产生与上层应用出现内存泄露而导致系统内存不足的可能性关系不大。

#### ■ 问题分析

为查明内存耗尽的真实原因，对单板启动这段时间内的内存信息进行查看：

1. 启动单板，应用程序全部运行并初始化完成，可以看到内存主要数据如下：

MemTotal:	2065864 kB	
MemFree:	10644 kB	
Cached:	1290588 kB	cached 近 1.3G，其中内存文件系统占用 207M，其它均为页缓存

HighTotal:	1310720 kB	
HighFree:	2924 kB	高端内存只剩 2.9M 左右
LowTotal:	755144 kB	
LowFree:	7720 kB	低端内存只剩 7.7M 左右

2. 运行应用一测试程序，将较快导致内存耗尽，此时内核打印信息如下：

```
DMA free:2600kB min:3524kB low:4404kB high:5284kB active:668052kB inactive:32540kB
present:777216kB pages_scanned:1950393 all_unreclaimable? yes
lowmem_reserve[]: 0 0 1265
HighMem free:1087180kB min:512kB low:1980kB high:3448kB active:205508kB inactive:16152kB
present:1295360kB pages_scanned:0 all_unreclaimable? no
lowmem_reserve[]: 0 0 0
```

由此可见，即使高端内存有剩余，但很多内核的内存申请和中断内存申请仅能面向已经内存不足的低端内存，且从以上信息可知，当内存不足时，低端内存区的内存已经不能再释放，而能进行释放的页缓存大多存在于高端内存中。

经过以上分析可以得出结论：由于高端内存被页缓存占据，迫使用户态应用程序只能从低端内存区中获取内存，同时，对只能从低端内存区分配内存的内核内存请求造成压力，因而低端内存的不足直接导致了系统内存不足。

内核本身的内存分配机制是存在一些问题的，在分配高端内存时，若高端不够用，会自动转向低端页面，但若转向从低端页面能够提供分配，就不会去启动内存回收机制，导致低端内存不断消耗，直到低端内存低于 low 水线。而此时再去启动，则为时已晚，是由于之前在低端分配均采用匿名映射，无法进行回收，致使真正需要低端内存的无法得到分配，从而出现问题。在服务器、桌面环境等非实时应用场景下，由于存在交换（swap）分区，因而可避免此问题。

### 2.12.2. 修改原理和方法

根据以上分析，该问题是由于页缓存无限制的分配所造成的。因而只要对页缓存大小进行限制，即可解决该故障。但考虑到今后对此类似问题有较好的跟踪、定位和处理，针对以下三部分分别提供了修改方案：

#### 1. 对页缓存大小的限制机制

内核提供对磁盘访问过程中产生的页缓存的限制功能，并且向应用提供接口以便该限制值可配置。

#### 2. 对内存使用情况的汇总显示



分别按照系统、内核、用户态应用程序三个层次来显示内存使用情况，当出现内存问题时，方便进行定位。

### 3. 在内存紧张时的处理机制

内存紧张时，内核会产生 OOM，造成系统运行的不确定性。通过使用水线预警和内存无法分配时的挂起机制，避免 OOM 的产生。同时，提供内存紧张时对特殊线程（管理、shell 等）的优先供给机制，确保管理功能正常工作。

## ■ 对页缓存的限制

内核的页缓存（page cache）主要由三部分构成：

- 磁盘访问时，内核为了提高访问速度，为磁盘文件分配、映射内存页面，从而对文件的读写实际为对这些内存页面的读写。内核负责对脏页面（缓存的内存页面内容与磁盘文件不一致）的回写（写入磁盘），同时当物理内存紧张时，内核将回收这些页缓存；
- 内存文件系统（ramfs）占用的页缓存，这些页缓存不能在内存紧张时被回收，除非对应的内存文件系统内的文件被删除；
- 使用 tmpfs 创建的共享内存，这一部分也不能被回收，除非应对的共享内存被删除。因此要达到对这些缓存的限制，必然涉及到对这些不同类型的缓存进行统计和限制两个部分。

### ◇ 页缓存统计

内核中对页缓存的页面数已经做了统计，从以上分析可知，我们目前的目标为仅限制磁盘访问时的页缓存，因此，需要将 ramfs 及 tmpfs 占用的页缓存从这个统计值中排除，这就涉及到对这两个部分占用页缓存的统计，实现方式为：

#### 1. 对 ramfs 的统计

- a) 增加 GFP 标志中增加一个 \_\_GFP\_PAGERAMFS 标志位，当 ramfs 创建本文件系统的 inode 节点时，对该 inode 节点的 mapping 设置 \_\_GFP\_PAGERAMFS 标志；
- b) 在内存区统计中增加 NR\_FILE\_RAMFS 项；
- c) 在内核现有对页缓存的 NR\_FILE\_PAGES 统计项进行增加或者减少的流程中，加入对页面 mapping 标志的判别，如果标志中 \_\_GFP\_PAGERAMFS 被置位，则说明该页缓存是属于 ramfs 的分配，于是将 NR\_FILE\_RAMFS 统计项做相应的增加或者减少，例如：

```
void __remove_from_page_cache(struct page *page)
{
    ... ..
    __dec_zone_page_state(page, NR_FILE_PAGES);
}
```

```

if(mapping_gfp_mask(mapping) & __GFP_PAGERAMFS)
    __dec_zone_page_state(page, NR_FILE_RAMFS);
else if(mapping_gfp_mask(mapping) & __GFP_PAGETMPFS)
    __dec_zone_page_state(page, NR_FILE_TMPFS);
... ..
}

```

## 2. tmpfs 的处理方法与 ramfs 类似。

经过以上处理，我们将页缓存的三部分分开，ramfs 与 tmpfs 分配由 NR\_FILE\_RAMFS 及 NR\_FILE\_TMPFS 对应的统计项描述，此时 NR\_FILE\_PAGES-NR\_FILE\_RAMFS-NR\_FILE\_TMPFS 便是用于磁盘访问时产生的页缓存统计值。

### ◇ 页缓存的限制

内核为页缓存分配物理页面的入口为 page\_cache\_alloc 以及 page\_cache\_alloc\_cold，为了使下游的物理页面分配流程识别这个分配，在 GFP 标志中增加 \_\_GFP\_PAGECACHE 标志位，并在调用以上两个函数时传入这个标志，修改点如下：

```

static inline struct page *page_cache_alloc(struct address_space *x)
{
    /*
     * add __GFP_PAGECACHE for page cache allocation expect page caches of
     * ramfs/tmpfs, so we can limit the cache size in alloc_pages.
     * xue.zhihong@20090729
     */
    if(!(mapping_gfp_mask(x) & (__GFP_PAGERAMFS | __GFP_PAGETMPFS)))
        return __page_cache_alloc(mapping_gfp_mask(x) | __GFP_PAGECACHE);
    else
        return __page_cache_alloc(mapping_gfp_mask(x));
}

```

Linux 系统所有物理页面的分配，其核心函数为 \_\_alloc\_pages，\_\_alloc\_pages 调用 get\_page\_from\_freelist 从各个内存区中分配物理页面，若函数中标志 \_\_GFP\_PAGECACHE 被置位，则表示这个请求是来自于页缓存的分配。因而此处采用统计值进行判别，即当已经分配的、用于磁盘访问的页缓存到达限制值时，则不允许继续进行分配，即使此时还有空闲内存。修改点如下：

```

static struct page *
get_page_from_freelist(gfp_t gfp_mask, unsigned int order,

```

```
struct zonelist *zonelist, int alloc_flags)
{
    ... ..
    if ((gfp_mask & __GFP_PAGECACHE) &&
        (zone_page_state(zone, NR_FILE_PAGES) -
         zone_page_state(zone, NR_FILE_RAMFS) -
         zone_page_state(zone, NR_FILE_TMPFS)) >
        zone->max_pagecache_pages){
        atomic_inc(&zone->cachefailed_times);
        goto try_next_zone;
    }
    ... ..
}
```

#### ◇ 用户接口

由于不同系统、不同设备对磁盘操作频度和方式不尽相同，所以必须对页缓存的限制值设置为可配置状态。因此，内核在 `proc` 文件系统中向应用提供配置这个限制值的方法：

##### 1. 向文件：

```
/proc/sys/vm/pagecache_ratio
```

##### 2. 写入数字以修改页缓存占总物理内存的比例（百分比），取值为：

$5 \leq \text{pagecache\_ratio} \leq 100$

## ■ 内存使用情况统计显示

[参考 2.11。](#)

## ■ 内存紧张时处理机制

在内存出现耗尽情况时，我们希望系统能以一种从容的方式来处理这种情况，能让应用得知这种情况的发生，并有机会采取相应措施。

这需要内核提供以下的支持：

##### 1. 在出现该情况时，内核应有通知机制，通知预定义的处理程序。

##### 2. 在出现内存耗尽时，系统不应任意杀死进程，出现不可控的行为，也不应该出现过多的打印，

占据大量 CPU，导致出现看门狗复位。

3. 内核应尽量保证处理程序的内存分配，以使处理动作能得到执行。

标准内核在系统内存紧张时的总体处理方式：如果通过分配标志判别到这个分配不能等待时，打印内存不足相关信息，其中包括堆栈回溯及各内存区的详细信息，之后返回空。如果可以等待，并且 `__GFP_FS` 被置位、`__GFP_NORETRY` 未被置位，则内核进入 OOM (Out Of Memory) 流程，这个流程将根据特定的策略寻找用户态进程杀死，以期进程的退出可释放内存，但是这个流程不能满足上面我们提出的要求，因此，对这个流程进行修改，总体处理方式为：

当内存不足时：

1. 内核发送信号通知上层应用 (SIGURG)，这样就给予上层应用对“系统内存紧张”这个事件的“知情权”，应用程序接收到这个信号之后，可以进行内存的释放、系统状态的记录、各进程内存占用情况的记录等等。同时，需要注意的是，由于此时内存压力已经很大，应用处理这个信号的流程需要避免各种方式的动态内存申请。这个流程涉及到用户接口为：

写入：

```
/proc/sys/vm/oom_handleurg_process
```

用以设置内核向哪个进程发送这个内存不足的信号，如果为 0，则内核此时向所有进程发送此信号。默认值为 0。进程如果没有注册对该信号的处理，那么默认动作是忽略该信号。

2. 如果是特殊线程（比如管理进程的关键线程、shell 代理线程），则此时允许该线程在水线以下分配内存，以便最大程度的保证内存紧张时这些线程能够运行，如果实在分配不到，则返回空，不挂起线程；而其他线程在内存不足时来申请内存，将被挂起一定时间，之后尝试第二次分配，如果第二次还是分配不到便挂起（线程进入 UNINTERRUPTIBLE 状态）。需要说明的是，虽然内核允许特殊线程在水线以下分配内存，但是水线以下内存也是很有限的，内核在内存压力发生时，并不能保证满足特殊线程的所有内存申请，同时水线以下内存的申请也需谨慎，因此应用设置哪些线程为特殊线程，也需谨慎，并且尽量使特殊线程的执行流程（例如 shell 代理线程的命令解析及执行）使用已经分配好的内存，而不进行过多的动态内存申请，这样与内核“尽力分配”结合起来，两方面共同作用以使得内存紧张时，系统的关键线程能够顺畅运行。

这个流程涉及到的用户接口为：

写入：

```
/proc/pid/oom_adj
```

如果线程对应的这个 proc 文件被写入 -17，则该线程便成为以上提到的特殊线程；

```
/proc/sys/vm/oom_wait_jiffies
```

用以设置非特殊线程在尝试第二次分配之前等待的时间，单位为系统 1/HZ，默认值为系统 HZ。

3. 修改内存不足时的打印，只打印必要的信息，避免嵌入式设备中经常遇到的问题：如果此时是在中断中分配，则过多的打印信息很容易造成看门狗溢出复位。

### 2.12.3. 使用方法

本补丁直接打入内核，无需进行配置。

### 2.12.4. 对标准内核影响

由于本补丁是针对标准内核的瑕疵进行的修复、完善，因此没有通过配置宏进行控制，而是直接将补丁功能打入内核，但已经过缜密的测试、验证，补丁是安全可靠的，已稳定发布。

## 2.13. Linux 页面分配及回收

### 2.13.1. 修改原因

#### ■ 问题描述

作为操作系统最重要组成部分的内存管理，在 Linux 中是以页为单位对内存进行管理的，即是说无论是分配还是释放内存都是按照页的整数倍进行的。为了提高性能，同时扩大物理内存的使用，Linux 采用了页缓存和交换的方式来实现内存的动态分配。

在 Linux 操作系统完成启动后，所有页的分配都是在 `__alloc_pages` 中完成的，页的释放是在 `__free_pages` 中完成。但在页分配的过程中会出现内存紧张的情况，这时就需要考虑对动态内存进行回收，以满足当前的需要。

#### ■ 问题分析

首先需要对内存管理区、内存页面分配流程有一个清晰的认识。

##### ➤ 内存管理区

Linux 以页为单位对内存进行管理，但在一些硬件体系架构上有的设备或者 CPU 不能访问高端的物理内存，因此 Linux 又按照物理地址从低到高的顺序组织了多个管理区。对于用户进程的页面申请，首先从高端内存开始申请，不能满足条件的再依次往下申请；对于有特殊需求的页面申请，可以指定所分

GDLC 版权所有 不得外传

配的管理区。下图列出了各个管理区的每个 CPU 页框缓存，及各个管理区的伙伴系统。

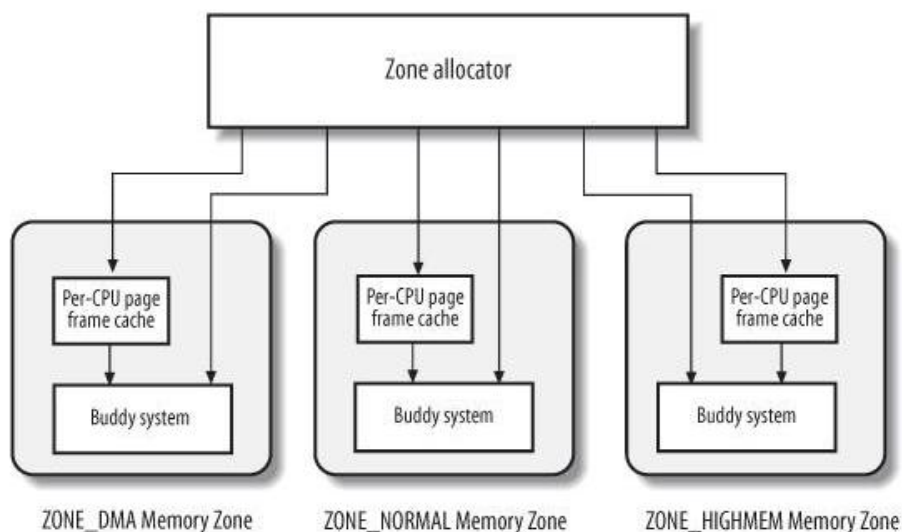


图 2-9 Linux 内存管理区组织结构

➤ 页面分配流程

分配页框主要通过调用 `alloc_pages` 函数，遍历当前系统的内存管理区获得满足条件的页框，同时搜集到内存分配失败的原因。其中，传入参数 `gfp_mask` 一组分配标志；`order` 以 2 的 `order` 次幂分配连续页框数量；`*zonelist` 管理区列表。

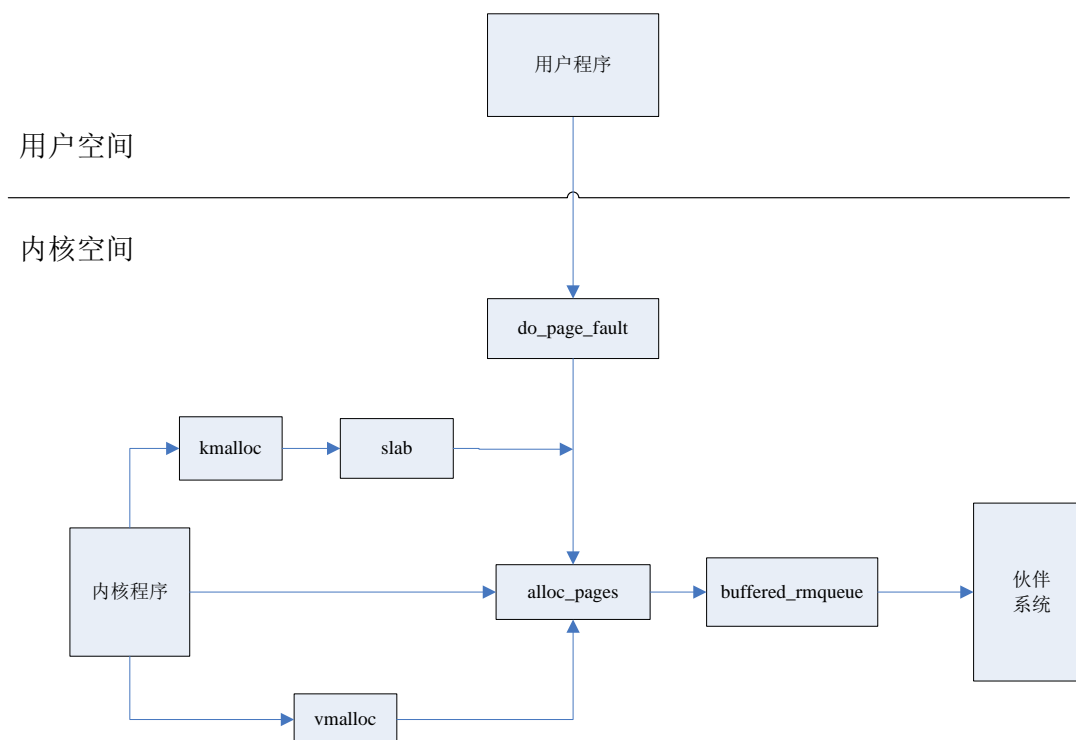


图 2-10 内存页面分配情况

## ✧ 页面分配顺序（管理区）

- a) 当 `gfp_mask & __GFP_DMA`，表示只能从 DMA 中获取页框；
- b) 当 `gfp_mask & __GFP_HIGHMEM`，表示按优先级从 HIGHMEM 到 NORMAL 再到 DMA 的顺序获取页框；
- c) 既无 `__GFP_DMA`，也没有 `__GFP_HIGHMEM`，表示按优先级从 NORMAL 到 DMA 的顺序获取页框；

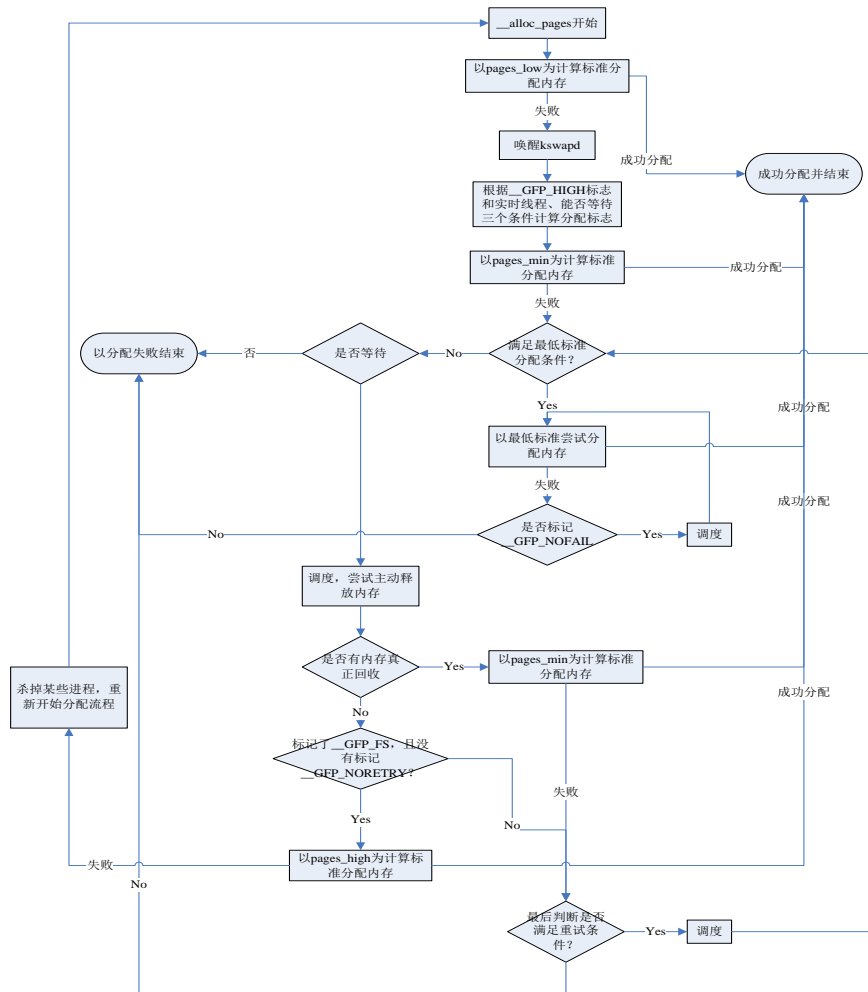
## ✧ 控制分配的关键字段

每个内存管理区的管理区描述符中均有三个关键字段：

- a) `pages_min`：表示管理区保留页框数
- b) `pages_low`：表示回收页框的下限，简单来讲，当管理区 `free_pages` 字段小于该值，应启动页回收机制
- c) `pages_high`：表示回收页框的上界，简单来讲，当管理区 `free_pages` 字段大于该值，应停止页回收机制

三个字段的值在系统初始化过程中计算产生，可通过修改 `/proc/sys/vm/min_free_kbytes` 变量进行调节。

## ✧ 分配流程



1. 以 `pages_low` 为计算标准调用 `get_page_from_freelist` 函数争取找到满足条件的空闲页框

遍历 zonelist 中的所有管理区，一旦管理区中 free\_pages 小于 pages\_low（根据传入参数情况，该值可能会经过加工），则开始遍历下一管理区。否则，则开始从该管理区伙伴系统中正式分配页框，然后结束 alloc\_page 流程。

 提示：如果 order 为 0，实际是从每个 CPU 页框缓存中分配单独的页框。

若所有管理区都是 `free_pages` 小于 `pages_low` 的情况，则此次 `get_page_from_freelist` 未找到满足条件的页框，进而尝试唤醒 `kswapd` 准备页框回收。进入步骤 2。

2. 以 `pages_min` 为计算标准调用 `get_page_from_freelist` 函数，即降低标准再次尝试分配



遍历 zonelist 中的所有管理区，一旦管理区中 free\_pages 小于 pages\_min（根据传入参数情况，该值可能会经过加工），则开始遍历下一管理区。否则，则开始从该管理区伙伴系统中正式分配页框，然后结束 alloc\_page 流程。

若所有管理区都是 free\_pages 小于 pages\_min 的情况，则此次 get\_page\_from\_freelist 未找到满足条件的页框。这时通过判断如下条件：

```
if (((p->flags & PF_MEMALLOC) || unlikely(test_thread_flag(TIF_MEMDIE)))
    && !in_interrupt()) {
    if (!(gfp_mask & __GFP_NOMEMALLOC)) {
```

条件满足则进入步骤 3。

条件不满足，且 (gfp\_mask & \_\_GFP\_WAIT)，则进入步骤 4。

其它情况以分配失败结束 alloc\_page 流程。

3. 以最低限度调用 get\_page\_from\_freelist 函数，即舍弃 pages\_min 和 pages\_low 的限制

遍历 zonelist 中的所有管理区，所遍历的管理区中没能直接分配到页框，则开始遍历下一管理区。

若有管理区满足分配，则开始从该管理区伙伴系统中正式分配页框，然后结束 alloc\_page 流程。

若仍没找到，这个时候就只有两条路：

- a) 以分配失败方式结束 alloc\_page 流程；
- b) 睡眠一段时间再进入第 3 步。依据条件如下：

```
if (gfp_mask & __GFP_NOFAIL)
```

4. 开始直接进行内存回收

调用 try\_to\_free\_pages 开始主动尝试内存回收，根据返回值判断，若有内存回收则再次以 pages\_min 为计算标准调用 get\_page\_from\_freelist 分配内存；如果没有内存回收，则进入步骤 5。

如果 get\_page\_from\_freelist 分配到了页面，则正常结束 alloc\_pages 流程。否则进入步骤 6。

5. 经过如下判断：

```
if ((gfp_mask & __GFP_FS) && !(gfp_mask & __GFP_NORETRY))
```

不满足条件则进入步骤 6

满足条件则以最高标准 pages\_high 为计算标准，尝试最后一次调用 get\_page\_from\_freelist，成功则结束 alloc\_pages 分配流程，否则则以内存耗尽为由开始有选择性的杀掉进程。由于杀掉了某些进程，因此应该会释放部分内存，此次分配也将回到初始，即步骤 1 重新开始分配流程。

6. 本步骤为 alloc\_pages 的最后一步，判断是否需要重试分配操作。如果不需要则将以内存分配失败结束 alloc\_pages 流程，否则会退回到步骤 2 的如下判断处：

```
if (((p->flags & PF_MEMALLOC) || unlikely(test_thread_flag(TIF_MEMDIE)))
    && !in_interrupt()) {
    if (!(gfp_mask & __GFP_NOMEMALLOC)) {
```

需要重试条件如下：

gfp\_mask 没有标记 \_\_GFP\_NORETRY，但标记了 \_\_GFP\_NOFAIL：不重试

gfp\_mask 没有标记 \_\_GFP\_NORETRY，但标记了 \_\_GFP\_REPEAT：不重试

gfp\_mask 没有标记 \_\_GFP\_NORETRY，且 order 小于等于 3：不重试

### 2.13.2. 修改原理和方法

分析了内存页面分配流程，则可以实施页面的回收操作。

#### ■ 内存回收方式

主要分为以下三种：

1. 紧缺内存回收：在内存分配时，发现内存不足即主动回收部分内存，接着再尝试分配；
2. 睡眠回收：进入 suspend-to-disk 状态，内核必须释放内存；
3. 周期回收：周期性激活内核线程执行内存回收算法。

函数调用关系如下图：

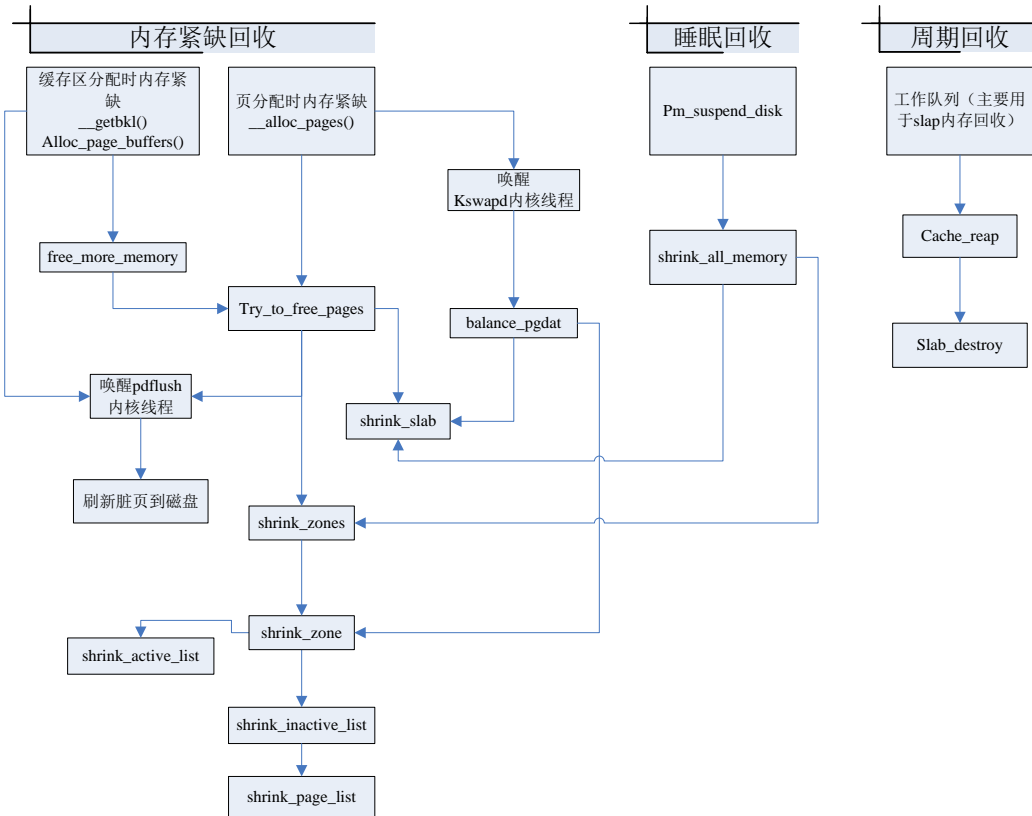


图 2-12 回收函数调用关系

## 回收目标及来源

回收目标：主要针对用户进程、页缓存和 slab 占用的内存。

与回收相关的用户进程和页缓存可由各内存管理区的 active 和 inactive 链表涵盖，slab 则以通过 set\_shrinker 的记录方式涵盖。

因此内存回收主要围绕 active、inactive 链表及 slab 进行，重点函数 shrink\_slab、shrink\_active\_list 和 shrink\_inactive\_list。

## 回收策略

1. 按照优先级从低到高进行扫描，优先级扫描的作用是控制每次扫描的页面个数，避免每次扫描所有的页面；
2. 一次扫描中从 Cache 和 slab 回收的总量达到 32 个页面即可认为回收成功，不再进行下面的扫描；
3. 如果一次扫描回收的页面数小于 32，那么判断本次扫描的 inactive 链表中的页面数是否超过 46，如果超过则唤醒 pdflush 线程。由于不能从 cache 中回收，有可能是页面中的内容需要写回文件，

所以这时可以唤醒 `pdflush` 线程进行写回操作；

4. 如果所有级别的扫描都已完成，即使回收的页面总数达不到 32，但也返回成功。剩余页面数是否足够在 `__alloc_pages` 中判断。

## ■ 优先级扫描算法

`Active` 和 `inactive` 链表中的页面个数未知，并且两个链表中的页面数也不会相同。从效率上考虑，对全链表进行扫描是不可行的，最好的方式是按页面数量由多到少依次扫描。

- 如果 `active` 中的页面数多，`inactive` 中的页面数少，那么就应该尽量对 `active` 链表进行扫描，以期释放更多页面；
- 如果 `active` 中的页面数少，`inactive` 中的页面数多，那么就尽量对 `inactive` 链表扫描，以期释放更多页面。

页面回收算法中对链表的扫描是按照 12 到 0 的优先级进行扫描的，每一个优先级针对链表中的页面数计算出不同的扫描数量。公式如下：

$$(\text{zone\_page\_state}(\text{zone}, \text{NR\_ACTIVE}) \gg \text{priority}) + 1 \geq 32$$

- 如果根据本优先级计算出的扫描数量不足 32，那么则不进行扫描，并将本次的扫描数量累计到下一优先级，这样可以使扫描尽快启动。
- 如果本优先级计算出的扫描数量超过 32，则立即进行扫描，并不累计到下一优先级。这样可以控制每次扫描的页面数。

那么我们根据这个计算公式可以看出链表中的页面数量要达到优先级的扫描条件，有如下的对应关系。

----->高							
12	11	10	9	8	7	.....	0
496M	240M	116M	56M	27M	13M	.....	0

- **shrink\_active\_list**

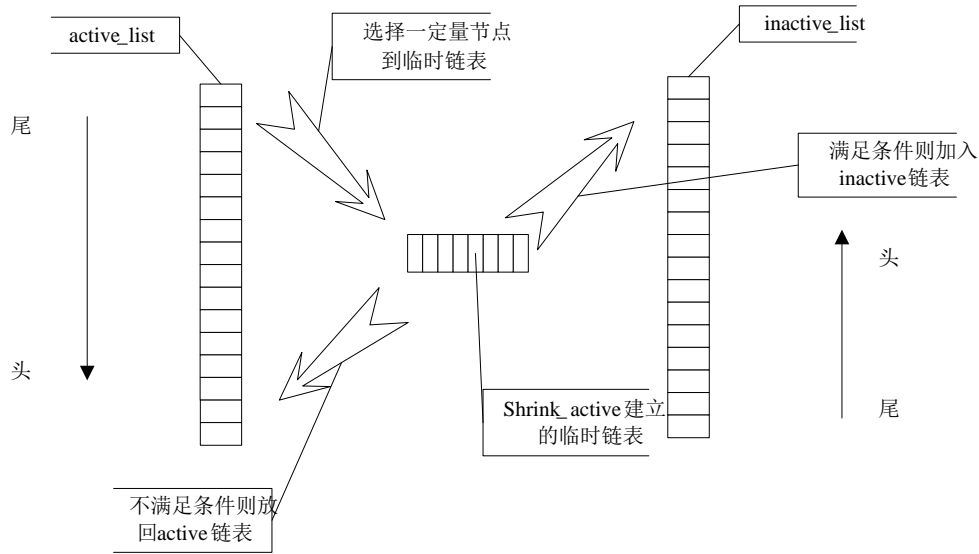


图 2-13 shrink\_active\_list 函数实现

## 1. 从 active 链表到临时链表

根据前面优先级扫描算法得到的页面数量 `nr_pages`, 从 active 链表的队尾开始取页面, 最大取 `nr_pages` 个放到临时链表中。

## 2. 从临时链表到 active 链表

如果页面正在被进程映射, 那么这个页面首先应考虑放回到 active 链表中, 根据 `page` 结构中的 `_mapcount` 字段可以判断页面是否被引用。放回 active 链表时, 是放到链表的队头。

## 3. 从临时链表到 inactive 链表

a) 从临时链表放到 inactive 链表时也是放到链表的队头;

b) 如果页面没被进程映射, 那么该页面就会被放到 inactive 链表中;

c) 同时, 对于内存紧张的情况, 即使该页面被映射也可以考虑回收或交换到磁盘中去。在以下几种情况都满足的条件下, 就该把页面放到 inactive 链表中:

✧ 当前页面回收困难, 满足以下任一条件均可认为回收已处于困难状态:

- ✓ 已扫描页面数超过 active 和 inactive 页面总数的 3 倍, 还没有页面被回收;
- ✓ 交换趋向值: 该值用于确定是移动所有的页还是只移动没被映射的页。当趋向值 `swap_tendency` 大于等于 100 的时候就认为该移动被映射的页面。

```
swap_tendency = mapped_ratio / 2 + distress + sc->swappiness;
```

其中:

```
mapped_ratio= ((global_page_state(NR_FILE_MAPPED) +global_page_state(NR_ANON_PAGES)) *
```

```
100) / vm_total_pages;
```

映射比例 mapped\_ratio 越大，说明 cache 中大部分被映射到用户态进程。

```
distress = 100 >> min(zone->prev_priority, priority);
```

负荷值 distress 是随优先级而改变的，优先级越小负荷值越大。

交换值 sc->swappiness 是一个用户定义常数，通常为 60。可以在 proc/sys/vm/swappiness 文件内修改。

- ✓ 对于匿名映射页面，只有在交换打开的情况下才放到 inactive 链表中；
- ✓ 对于文件映射页面，只有在最近没被引用过并且虚拟地址没加 VM\_LOCKED 标志的条件下，才放到 inactive 链表中。

#### ➤ shrink\_inactive\_list

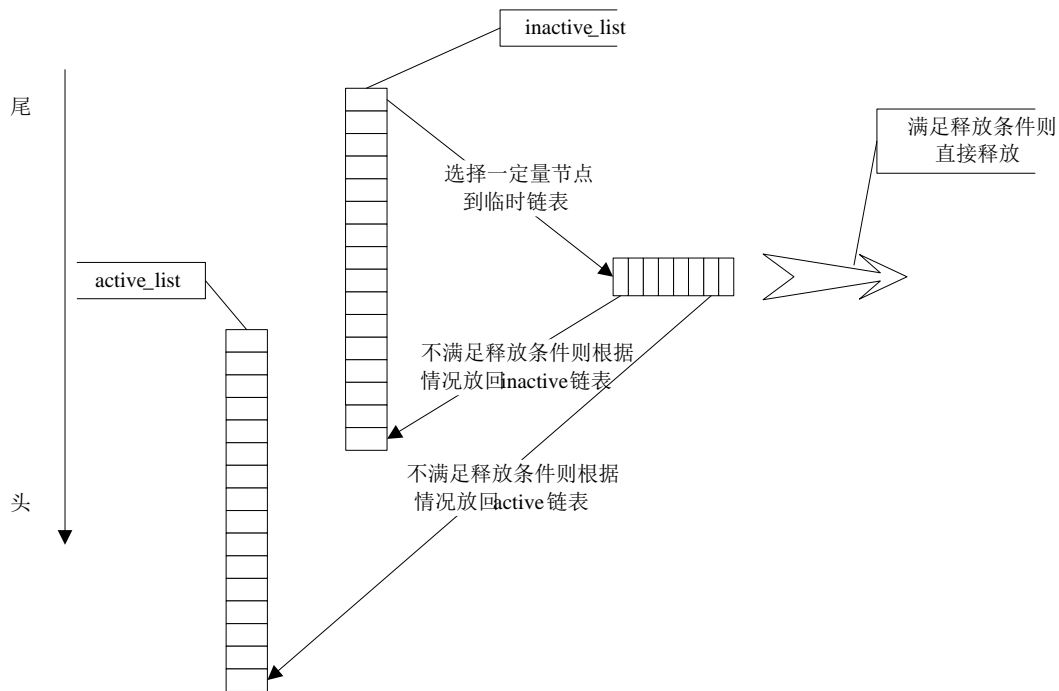


图 2-14 shrink\_inactive\_list 函数实现

##### 1. 从 inactive 链表到临时链表

根据前面优先级扫描算法得到的页面数量 nr\_pages，从 inactive 链表的队尾开始取页面，最大取 nr\_pages 个放到临时链表中。

##### 2. 从临时链表到 active 链表

调用 `shrink_page_list` 对临时链表进行回收操作，回收完成后如果临时链表中的页面被置为了 PG\_active，则重新放回到 active 链表中；

### 3. 从临时链表到 inactive 链表

调用 `shrink_page_list` 对临时链表进行回收操作，回收完成后如果临时链表中的页面没有被置为了 `PG_active`，则重新放到 `inactive` 链表中；

### 4. `shrink_page_list` 对临时链表的回收

- a) 如果页面被置了 `PG_locked` 标志，则不回收页面，仍保持 `inactive` 状态；
- b) 如果页面被置了 `PG_writeback` 标志，表示正在回写，继续等待回写完成；
- c) 根据 LRU 算法（见后）判断页面重新被引用过，则将页面置为 `PG_active`；
- d) 如果是匿名映射页面，且 `swap` 打开，那么则尝试将页面交换到磁盘中；如果页面交换失败，则重新将页面置为 `PG_active`；
- e) 如果页面正被进程映射（匿名和文件），且能进行反向映射（`page->mapping`），那么尝试对各个映射进程的页表项进行解除映射；如果解除映射失败，那么可以重新将页面置为 `PG_active`；
- f) 如果页面是脏页，那么尝试将页面写回文件；如果写回失败，那么可以重新将页面置为 `PG_active`；
- g) 如果页面是块设备缓冲头部，那么尝试对整个块设备缓冲进行释放；如果释放失败，那么重新将页面置为 `PG_active`；
- h) 如果页面正被进程映射，但无法进行反向映射（`page->mapping` 为 `NULL`），那么将页面仍然保留在 `inactive` 链表中；

## ■ 页面最近最少使用（LRU）算法

要能够判断页面最近是否被使用过，最好的方法是在页表项中增加一个对硬件的访问计数来表示这个页面最近是否被访问过，但是目前只有很少的 CPU 支持这一特性。那么就只能通过软件模拟的方法来实现这一功能，实现的方法是通过页框中的 `PG_referenced` 和 `PG_active` 两个标志的组合表示页面最近是否被访问过，仅用一个标志不足以表达页面状态的变化过程。

- `PG_active`：表示页面是否处于激活状态，处于激活状态则应该放到 `active` 链表中；
- `PG_referenced`：表示页面是否被引用过；

下图表示了页面状态的转换过程：

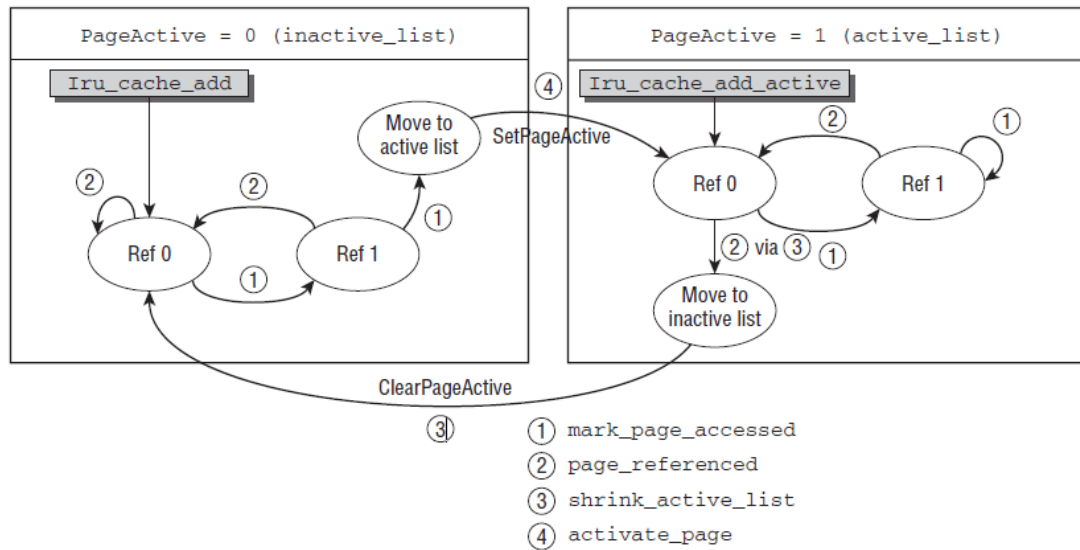


图 2-15 页面状态转换

其中：

active\_list 中的状态转换

1. Ref 1 : PG\_active=1, PG\_referenced=1
2. Ref 0 : PG\_active=1, PG\_referenced=0

inactive\_list 中的状态转换

1. Ref 1 : PG\_active=0, PG\_referenced=1
2. Ref 0 : PG\_active=0, PG\_referenced=0

mark\_page\_accessed 对 PG\_referenced 和 PG\_active 标志有影响；

page\_referenced 对 PG\_referenced 标志有影响。

## ■ shrink\_slab

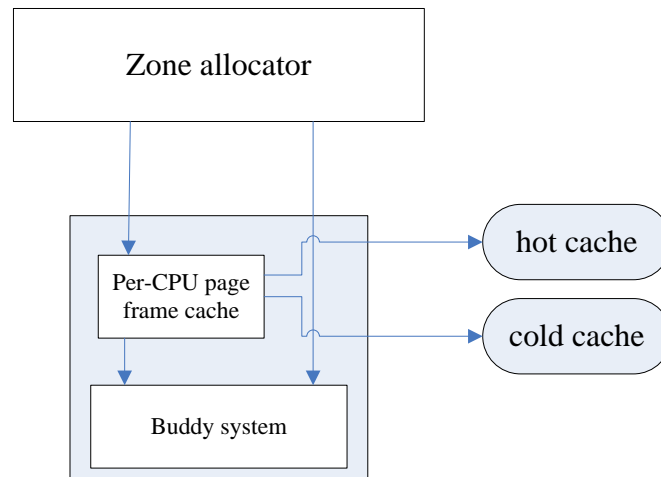
shrink\_slab 主要针对以 set\_shrinker 向 PFRA 注册的几种专用 slab 高速缓存，遍历 shrinker\_list 指向的链表。

向 PFRA 注册的 slab 很少，除了目录项高速缓存和索引节点高速缓存，还有磁盘限额层、文件系统元信息块高速缓存和 XFS 日志文件系统。



## ■ 页面释放

1. 根据 page 结构的 `_count` 字段判断页面是否能被释放，因为这个页面可能被多个进程映射，或者在内核某些临界区域中有意将页面的 `_count` 计数增加，使该页面不能被释放；
2. 判断被释放的是否仅为一个页面。如果是则调用 `free_hot_page` 将页面释放到每 CPU 的热缓存中。  
每 CPU 页框缓存分为两个链表：一个是热缓存，一个是冷缓存。



- 页面释放到 CPU 缓存时都是释放到热缓存；
  - 如果热缓存的页面数量达到 `high` 值时，则触发 `free_pages_bulk` 将热缓存队尾的页面释放；
  - 如果热缓存的页面数量小于 `high` 值时，则将刚释放的页面放入热缓存队头，并将 `count` 计数加 1。
3. 如果被释放的页面数大于 1，那么则调用 `__free_pages_ok` 接口将连续的页面释放回伙伴系统。在释放前判断一下这些页面中有没有 `reserve` 标志被置位。

### 2.13.3. 使用方法

本功能为标准 Linux 自带功能。

### 2.13.4. 对标准内核影响

此项功能为标准 Linux 自带，已稳定发布。

## 2.14. 程序代码段、数据段巨页映射

### 2.14.1. 修改原因

#### ■ 问题描述

中研某项目运行过程中，发现媒体面进程性能不能满足需要。经分析发现是由于内核中产生了大量的 tlb miss 异常所致。媒体面进程通过 mmap 映射了约 1.5G 共享内存，应用程序频繁访问这些共享内存，造成不断产生 tlb miss 异常。将这些共享内存的页面从 16K 修改为 128M 巨页映射后，系统性能大幅提高。由于应用程序的代码段、数据段也会造成 tlb miss 异常，因此为了进一步提升系统性能，决定实现应用程序代码段、数据段巨页映射功能。最初代码段和数据段的大小限制为 32M，即内核需要实现“32M 代码数据段巨页映射功能”。

后来项目在使用过程中发现，随着平台版本代码的增加，代码段已经超过了 32M 的限制，需要内核实现支持“64M 代码数据段巨页映射功能”。为了不影响原有 32M 功能，需要做到 32M&64M 代码数据段巨页映射功能兼容。

#### ■ 问题分析

项目使用的单板是“MIPS XLR 732”64 位单板，共 8 个物理 CPU，32 个逻辑 CPU 核。每个逻辑核只有 16 个 tlb entry。一个 tlb entry 可以映射 16K 页面，这，只要连续访问  $16 * 16 = 256K$  范围以上的数据，就必然会造成 tlb miss 异常。而应用程序的逻辑地址空间是 2G，因此很容易造成 tlb miss。曾经尝试在关键函数中调用了空函数，发现系统性能急剧下降，这也就成为要求实现代码段、数据段巨页映射的直接原因。为实现此功能需要 TSP 提供定制的连接脚本，使代码段和数据段按照 32M 或者 64M 对齐。

#### ➤ 32M 代码数据段巨页功能

代码段和数据段各占用 32M 内存，如图 2-16 所示 EntryHi 表示虚拟地址，EntryLo0 映射代码段的前 16M，而 EntryLo1 映射代码段的后 16M。

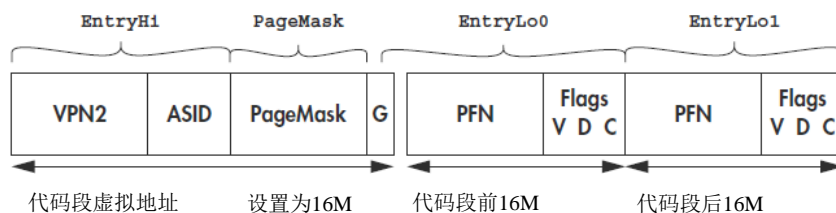


图 2-16 TLB entry 映射 (32M)

➤ 64M 代码数据段巨页功能

代码段和数据段各占用 64M，如下图所示 EntryHi 表示虚拟地址，EntryLo0 用于映射 64M 代码段，而 EntryLo1 用于映射 64M 数据段。

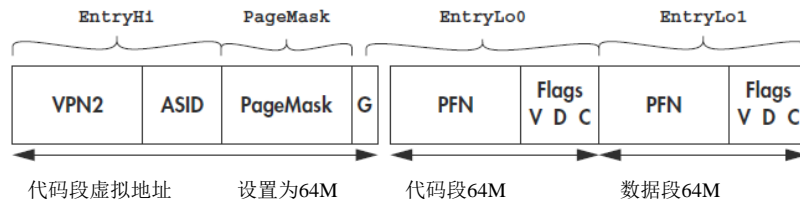


图 2-17 TLB entry 映射 (64M)

### 2.14.2. 修改原理和方法

在装载应用程序代码段和数据段时，将代码段和数据段的 `vm_area_struct` 结构设置上一个特殊标志：**VM\_ELF\_HUGETLB**。这个标志表示该应用程序的代码段、数据段需要进行巨页映射。如果应用程序不是媒体面进程或控制面进程，就不设置这个标志。当 `elf_mmap` 函数调用 `do_mmap_pgoff` 函数建立数据段和代码段的 `vm_area_struct` 时，为数据段和代码分配好连续的物理内存块，并从可执行程序文件中读取代码段、数据段的内容到物理内存块中。同时，如果代码段、数据段进行了巨页映射，就不再进行内存即时映射。这是因为内存即时映射会从可执行程序的文件缓存中为代码段、数据段建立 16K 页面映射。这是需要避免的。

当可执行程序访问代码段、数据段时，会产生 `tlb miss` 异常，由于我们没有为代码段、数据段建立任何 `pte` 页表项，因此，`tlb miss` 异常最终会调用 `do_page_fault` 函数进行缺页处理。为此，我们修改 `do_page_fault` 处理流程，当发现 `vm_area_struct` 结构有特殊标志 `VM_ELF_HUGETLB` 时，走巨页映射的缺页处理流程，而不再走原来的 `filemap_nopage` 流程。在巨页映射缺页处理流程中，直接操作 CPU 核的 `tlb entry` 进行巨页映射。

#### ■ 代码说明

修改的文件清单有：

- `arch\mips\mm\fault.c`
- `fs\binfmt_elf.c`
- `include\asm-mips\mman.h`
- `include\linux\mm.h`

➤ mm\mmap.c

下面对修改的代码进行逐一说明：

■ fs\binfmt\_elf.c 文件：

```
static int load_elf_binary(struct linux_binprm *bprm, struct pt_regs *regs)
{
    .....
    /**
     * 32M 对齐链接脚本含有特殊段名称为 “pageset”，而 64M 对齐链接脚本含有特殊段名称
     * 为 “64m_pageset”。使用全局变量 code_data_elf_hugetlb_type 表示内核启用 32M 巨页或
     * 64M 巨页功能，1 表示 32M 巨页功能，2 表示 64M 巨页功能。若两个进程使用的链接脚本类型不
     * 一致，内核会给出相应提示。为了避免两个进程同时走到判断流程出现逻辑问题，这里用信号
     * 量 elf_hugetlb_type_sem 来进行保护。
     */
    down(&elf_hugetlb_type_sem);
    if (1 == huge_tlb_type) {
        if (2 == code_data_elf_hugetlb_type) {
            printk(KERN_ERR "!!! 64m code&data hugetlb, linksript not match !!!\n");
            BUG();
        } else {
            code_data_elf_hugetlb_type = 1;
            printk("enable hugetlb for 32m code segment and data segment\n");
            bprm->interp_flags |= BINPRM_FLAGS_HUGETLB_ELF;
        }
    } else if (2 == huge_tlb_type) {
        if (1 == code_data_elf_hugetlb_type) {
            printk(KERN_ERR "!!! 32m code&data hugetlb, linksript not match !!!\n");
            BUG();
        } else {
            code_data_elf_hugetlb_type = 2;
            printk("enable hugetlb for 64m code segment and data segment\n");
            bprm->interp_flags |= BINPRM_FLAGS_HUGETLB_ELF;
        }
    }
    up(&elf_hugetlb_type_sem);
}
```

```
.....  
/**  
 * 堆空间按照 32M 或 64M 对齐  
 */  
if (bprm->interp_flags & BINPRM_FLAGS_HUGETLB_ELF)  
{  
    if (2 == code_data_elf_hugetlb_type) {  
        elf_bss = (elf_bss + 64 * 1024 * 1024 - 1) & ~(64 * 1024 * 1024 - 1);  
        elf_brk = (elf_brk + 64 * 1024 * 1024 - 1) & ~(64 * 1024 * 1024 - 1);  
    } else {  
        elf_bss = (elf_bss + 32 * 1024 * 1024 - 1) & ~(32 * 1024 * 1024 - 1);  
        elf_brk = (elf_brk + 32 * 1024 * 1024 - 1) & ~(32 * 1024 * 1024 - 1);  
    }  
    .....  
}
```

#### ■ mm/mmap.c 文件:

```
#ifdef CONFIG_HUGETLB_ELF  
extern unsigned long code_data_seg_hugetlb_mem;  
unsigned long elf_hugetlb_phy = 0;  
unsigned long elf_hugetlb_used[4] = {0};  
DECLARE_MUTEX(elf_hugetlb_sem);  
int code_data_elf_hugetlb_type = -1;  
void free_elf_hugetlb_seg(void *data, unsigned long len)  
{  
    int i;  
    for (i = 0; i < 4; i++)  
    {  
        if (elf_hugetlb_used[i] == (unsigned long)data)  
        {  
            elf_hugetlb_used[i] = 0;  
        }  
    }  
}  
/**
```

\* 这里本来应该使用 **BSP** 的接口来申请巨页内存的。但 **CGEL** 没有提供类似的接口，所以这里采用在启动阶段使用 **alloc\_bootm** 来预留相应的物理内存，指针 **code\_data\_seg\_hugetlb\_mem** 表示预留物理内存的起始地址。

\*/

```
unsigned long alloc_elf_hugetlb_seg(unsigned long len)
{
    unsigned long ret = 0, max_seg_size;
    int i;

    if (2 == code_data_elf_hugetlb_type) {
        max_seg_size = 64 * 1024 * 1024;
    } else {
        max_seg_size = 32 * 1024 * 1024;
    }

    if (elf_hugetlb_phy == 0)
    {
        int i;
        elf_hugetlb_phy = code_data_seg_hugetlb_mem;
        for (i = 0; i < 4; i++)
        {
            elf_hugetlb_used[i] = 0;
        }
    }

    BUG_ON(elf_hugetlb_phy == 0);
    ret = (unsigned long)__va(elf_hugetlb_phy);
    for (i = 0; i < 4; i++)
    {
        if (elf_hugetlb_used[i] == 0)
        {
            ret = elf_hugetlb_used[i] = (unsigned long)__va(elf_hugetlb_phy + max_seg_size * i);
            memset((void *)ret, 0, max_seg_size);
            return ret;
        }
    }
}
```

```
BUG();
return ret;
}
#endif

static struct vm_area_struct *remove_vma(struct vm_area_struct *vma)
{
    .....
    /**
     * 在应用程序调用 exit_mm 时，会调用 remove_vma，在此释放巨页分配的物理内存。
     * 虽然在 vma 合并时，也需要释放相关资源，但是巨页映射的段是不允许合并，因此仅仅在此释放就
     * 行了。
     */
    #ifdef CONFIG_HUGETLB_ELF
        if (vma->elf_file_data)
            free_elf_hugetlb_seg(vma->elf_file_data, vma->vm_end - vma->vm_start);
    #endif
    .....
}

static inline int is_mergeable_vma(struct vm_area_struct *vma,
                                   struct file *file, unsigned long vm_flags)
{
    .....
    /**
     * 巨页映射的段不能被合并。原因如上。
     */
    #ifdef CONFIG_HUGETLB_ELF
        if (vma->vm_flags & VM_ELF_HUGETLB)
            return 0;
    #endif
    return 1;
}

unsigned long do_mmap_pgoff(struct file * file, unsigned long addr,
                           unsigned long len, unsigned long prot,
                           unsigned long flags, unsigned long pgoff)
```

```
{
    .....
/**
 * 巨页映射的段，最小必须为 2M 或 4M。如果代码段、数据段低于 2M 或 4M，强制向上对齐。
 * 并且按照 4 的倍数递增，这主要是 MIPS 页面映射的要求。32M 功能最小按 2M 对齐，64M 功能最小
 * 按 4M 对齐。
 */
#ifdef CONFIG_HUGETLB_ELF
    int orig_len = len;
    if (flags & MAP_ELF_HUGETLB)
    {
        int tmp;

        if (1 == code_data_elf_hugetlb_type) {
            tmp = 1UL << 21;
        } else {
            tmp = 1UL << 22;
        }

        while (tmp < len)
        {
            tmp = tmp << 2;
        }

        len = tmp;
    }
#endif

/**
 * 如果传递了 MAP_ELF_HUGETLB 标志，就设置 vma 的 VM_ELF_HUGETLB 标志。
 * 请注意两个标志的不同之处。
 */
#ifdef CONFIG_HUGETLB_ELF
    if (flags & MAP_ELF_HUGETLB)
    {
        vm_flags |= VM_ELF_HUGETLB;
```



```
    }
#endif

/**
 * 只要有 MAP_ELF_HUGETLB 标志，就不能再设置 VM_ALWAYSLOCKED 标志。
 * 因为 VM_ALWAYSLOCKED 标志的 vma 会调用 make_pages_present 函数，为代码段和数据段
 * 建立 pte 页表项，导致巨页映射不能生效。
 */
#ifdef CONFIG_HUGETLB_ELF
    if (flags & MAP_ELF_HUGETLB)
    {
        vm_flags &= (~VM_ALWAYSLOCKED);
    }
#endif

/**
 * 下面这段代码从文件中读取代码段、数据段内容到物理内存中。
 * 并计算好 tlb 的属性
 */
#ifdef CONFIG_HUGETLB_ELF
    if (flags & MAP_ELF_HUGETLB)
    {
        int retval = 0;
        int shift = 0;
        unsigned long half_len = len >> 1;

        vma->elf_file_data = alloc_elf_hugetlb_seg(len);
        BUG_ON(vma->elf_file_data == NULL);

        vma->elf_vaddr = addr;
        vma->elf_phys0 = __pa(vma->elf_file_data);
        vma->elf_phys1 = vma->elf_phys0 + half_len;
        vma->elf_flags = 0x1e; //protection_map[vm_flags &
                               //(VM_READ|VM_WRITE|VM_EXEC)].pgprot;

        if (half_len == 1UL << 20)
```

```
{
    vma->elf_mask = 0x001fe000;
}
else if (half_len == 1UL << 21)
{
    vma->elf_mask = 0x003fe000;
}
else if (half_len == 1UL << 22)
{
    vma->elf_mask = 0x007fe000;
}
else if (half_len == 1UL << 23)
{
    vma->elf_mask = 0x00ffe000;
}
else if (half_len == 1UL << 24)
{
    vma->elf_mask = 0x01ffe000;
}
else if (half_len == 1UL << 25)
{
    vma->elf_mask = 0x03ffe000;
}
else if (half_len == 1UL << 26)
{
    vma->elf_mask = 0x07ffe000;
}
else
{
    BUG();
}
vma->elf_mask2 = ~(len - 1);

retval = kernel_read(file, pgoff << PAGE_SHIFT, vma->elf_file_data, orig_len);

if (retval != orig_len) {
```

```
        BUG();
    }
}
else
{
    vma->elf_file_data = NULL;
}
#endif
```

■ Arch/mips/mm/fault.c 文件:

```
asmlinkage void do_page_fault(struct pt_regs *regs, unsigned long write,
                             unsigned long address)
{
    .....
    /**
     * 如果是使用 32M 功能就调用函数 elf_set_huge_tlb()进行映射，如果是使用 64M 功能就调用函数
     * elf_set_huge_tlb_64m()进行映射
     */
    if (!mm)
        goto bad_area_nosemaphore;

#ifdef CONFIG_HUGETLB_ELF
    /**
     * xby for CGN, 2010-06-24
     */
    if (1 == code_data_elf_hugetlb_type) {
        if (address >= mm->vma_code_start &&
            address < mm->vma_code_end)
        {
            vma = mm->vma_code;
            elf_set_huge_tlb(vma);

            return;
        }
    }
}
```

```
if (address >= mm->vma_data_start &&
    address < mm->vma_data_end)
{
    vma = mm->vma_data;
    elf_set_huge_tlb(vma);

    return;
}
} else if (2 == code_data_elf_hugetlb_type) {
    if ((address >= mm->vma_code_start && address < mm->vma_code_end) ||
        (address >= mm->vma_data_start && address < mm->vma_data_end)) {

        elf_set_huge_tlb_64m(mm->vma_code, mm->vma_data);

        return;
    }
}
#endif

if (in_atomic())
    goto bad_area_nosemaphore;.....
.....
}
```

### 2.14.3. 使用方法

在 menuconfig 中打开 “**hugetlb for code segment and data segment ---> Enable hugetlb for code segment and data segment**” 功能选项。

同时，由于进行巨页映射时必须保证对应的虚拟地址是 2 的 N 次方。因此需要修改应用程序的链接脚本，保证代码段和数据段按照 32M 或 64M 进行对齐。应用程序使用 32M 对齐链接脚本进行链接，则内核会启用 32M 代码数据段巨页功能；应用程序使用 64M 对齐链接脚本进行链接，则内核会启用 64M 代码数据段巨页功能。

32M 对齐链接脚本如下，后者表示 BSS 段并入数据段。

#### ■ HugeMap32.lds

```

/* Script for -z combrelloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf64-tradbigmips", "elf64-tradbigmips",
              "elf64-tradlittlemips")
OUTPUT_ARCH(mips)
ENTRY(__start)
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib64");
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib");
SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x120000000)); . =
SEGMENT_START("text-segment", 0x120000000) + SIZEOF_HEADERS;
    .MIPS.options : { *(.MIPS.options) }
    .note.gnu.build-id : { *(.note.gnu.build-id) }
    .dynamic       : { *(.dynamic) }
    .hash          : { *(.hash) }
    .gnu.hash      : { *(.gnu.hash) }
    .dynsym        : { *(.dynsym) }
    .dynstr        : { *(.dynstr) }
    .gnu.version   : { *(.gnu.version) }
    .gnu.version_d : { *(.gnu.version_d) }
    .gnu.version_r : { *(.gnu.version_r) }
    .rel.dyn       :
    {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro* .rel.gnu.linkonce.d.rel.ro.*)
        *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
        *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
        *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
        *(.rel.ctors)
        *(.rel.dtors)
    }
}

```

```

*(.rel.got)
*(.rel.dyn)
*(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
*(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
*(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
*(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
*(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
PROVIDE_HIDDEN (__rel_iplt_start = .);
*(.rel.iplt)
PROVIDE_HIDDEN (__rel_iplt_end = .);
PROVIDE_HIDDEN (__rela_iplt_start = .);
PROVIDE_HIDDEN (__rela_iplt_end = .);
}
.rela.dyn :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
    PROVIDE_HIDDEN (__rel_iplt_start = .);
    PROVIDE_HIDDEN (__rel_iplt_end = .);
    PROVIDE_HIDDEN (__rela_iplt_start = .);
    *(.rela.iplt)
    PROVIDE_HIDDEN (__rela_iplt_end = .);
}

```

```
.rel.plt      :
{
    *(.rel.plt)
}
.rela.plt     :
{
    *(.rela.plt)
}
.init        :
{
    KEEP (*( .init))
} =0
.plt         : { *(.plt) }
.iplt        : { *(.iplt) }
.text        :
{
    _ftext = . ;
    *(.text.unlikely .text.*_unlikely)
    *(.text.exit .text.exit.*)
    *(.text.startup .text.startup.*)
    *(.text.hot .text.hot.*)
    *(.text.stub .text.*.gnu.linkonce.t.*)
    /* .gnu.warning sections are handled specially by elf32.em.  */
    *(.gnu.warning)
    *(.mips16.fn.*) *(.mips16.call.*)
} =0
.fini        :
{
    KEEP (*( .fini))
} =0
PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
.rodata      : { *(.rodata .rodata.*.gnu.linkonce.r.*) }
.rodata1     : { *(.rodata1) }
.sdata2      :
```

```

{
    *(.sdata2 .sdata2.* .gnu.linkonce.s2.*)
}

.sbss2          : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr  : { *(.eh_frame_hdr) }
.eh_frame       : ONLY_IF_RO { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RO { *(.gcc_except_table .gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (CONSTANT (MAXPAGESIZE)) - ((CONSTANT (MAXPAGESIZE) - .) & (CONSTANT
(MAXPAGESIZE) - 1)); . = DATA_SEGMENT_ALIGN (CONSTANT (MAXPAGESIZE), CONSTANT
(COMMONPAGESIZE));
/* Exception handling */
/* Modified by ZTE-OS:
   the .rodata section is the last one defaultly arranged into data segment in  XGW media/control
   process.
   We set the location counter "." to the next 32MB border,
   and the excutable segment thus occupies 32MB space in VMA*/
. = ALIGN(0x2000000);
.eh_frame       : ONLY_IF_RW { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RW { *(.gcc_except_table .gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata  : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss    : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
.preinit_array :
{
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);
}
.init_array :
{
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*(init_array))
    PROVIDE_HIDDEN (__init_array_end = .);
}

```



```

}
.fini_array      :
{
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP (*(SORT(.fini_array.*)))
    KEEP (*(.(fini_array)))
    PROVIDE_HIDDEN (__fini_array_end = .);
}
.ctors           :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin.o(.ctors))
    KEEP (*crtbegin?.o(.ctors))
    /* We don't want to include the .ctor section from
       the crtend.o file until after the sorted ctors.
       The .ctor section from the crtend file contains the
       end of ctors marker and it must be last */
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o ) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*(.(ctors)))
}
.dtors           :
{
    KEEP (*crtbegin.o(.dtors))
    KEEP (*crtbegin?.o(.dtors))
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o ) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*(.(dtors)))
}

```

```

}

.jcr          : { KEEP (*.jcr) }

.data.rel.ro : { (*.data.rel.ro.local* .gnu.linkonce.d.rel.ro.local.*) (*.data.rel.ro* .gnu.linkonce.d.rel.ro.*) }

. = DATA_SEGMENT_RELRO_END (0, .);

/* Modified by ZTE-OS:

   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.

   We set the location counter "." to the next 64MB border,

   and the data segment thus occupies 64MB space in VMA*/

.ps_spec_data : { (*.ps_spec_data)}

.pageset      :

{
    __HUGE_TLB_SIZE = . ;
    LONG(0x2000000)
}

.data          :

{
    _fdata = . ;
    (*.data .data.* .gnu.linkonce.d.*)
    SORT(CONSTRUCTORS)
}

.data1         : { (*.data1) }

.got.plt       : { (*.got.plt) }

. = .;

_gp = ALIGN(16) + 0x7ff0;

.got           : { (*.got) }

/* We want the small data sections together, so single-instruction offsets

   can access them all, and initialized data all before uninitialized, so

   we can shorten the on-disk segment size.  */

.sdata         :

{
    (*.sdata .sdata.* .gnu.linkonce.s.*)
}

.lit8          : { (*.lit8) }

.lit4          : { (*.lit4) }

.srdata        : { (*.srdata) }

_edata = .; PROVIDE (edata = .);

```

```

/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 32MB border,
   and the data segment thus occupies 32MB space in VMA*/
. = ALIGN(0x2000000);
__bss_start = .;
_fbss = .;
.sbss          :
{
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
}
.bss          :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
/* Align here to ensure that the .bss section occupies space up to
   _end.  Align after .bss to ensure correct alignment even if the
   .bss section disappears because there are no input sections.
   FIXME: Why do we need it? When there is no .bss section, we don't
   pad the .data section.  */
. = ALIGN(. != 0 ? 64 / 8 : 1);
}
. = ALIGN(64 / 8);
. = ALIGN(64 / 8);
_end = .; PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections.  */
.stab          0 : { *(.stab) }
.stabstr       0 : { *(.stabstr) }
.stab.excl     0 : { *(.stab.excl) }
.stab.exclstr  0 : { *(.stab.exclstr) }
.stab.index    0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }

```

```
.comment      0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0.  */
/* DWARF 1 */
.debug        0 : { *(.debug) }
.line         0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo .zdebug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames .zdebug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges .zdebug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames .zdebug_pubnames) }
/* DWARF 2 */
.debug_info    0 : { *(.debug_info .gnu.linkonce.wi.* .zdebug_info) }
.debug_abbrev   0 : { *(.debug_abbrev .zdebug_abbrev) }
.debug_line     0 : { *(.debug_line .zdebug_line) }
.debug_frame    0 : { *(.debug_frame .zdebug_frame) }
.debug_str      0 : { *(.debug_str .zdebug_str) }
.debug_loc      0 : { *(.debug_loc .zdebug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo .zdebug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames .zdebug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames .zdebug_funcnames) }
.debug_tynames  0 : { *(.debug_tynames .zdebug_tynames) }
.debug_varnames 0 : { *(.debug_varnames .zdebug_varnames) }
/* DWARF 3 */
.debug_pubtypes 0 : { *(.debug_pubtypes .zdebug_pubtypes) }
.debug_ranges   0 : { *(.debug_ranges .zdebug_ranges) }
.gnu.attributes 0 : { KEEP (*(.gnu.attributes)) }
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
/DISCARD/ : { *(.note.GNU-stack) *(.gnu_debuglink) *(.gnu.lto_*) }
}
```

```

/* Script for -z combrelloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf64-tradbigmips", "elf64-tradbigmips",
              "elf64-tradlitlemips")
OUTPUT_ARCH(mips)
ENTRY(__start)
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib64");
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib");
SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x120000000)); . =
SEGMENT_START("text-segment", 0x120000000) + SIZEOF_HEADERS;
    .MIPS.options : { *(.MIPS.options) }
    .note.gnu.build-id : { *(.note.gnu.build-id) }
    .dynamic       : { *(.dynamic) }
    .hash          : { *(.hash) }
    .gnu.hash      : { *(.gnu.hash) }
    .dynsym        : { *(.dynsym) }
    .dynstr        : { *(.dynstr) }
    .gnu.version   : { *(.gnu.version) }
    .gnu.version_d : { *(.gnu.version_d) }
    .gnu.version_r : { *(.gnu.version_r) }
    .rel.dyn       :
    {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro* .rel.gnu.linkonce.d.rel.ro.*)
        *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
        *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
        *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
        *(.rel.ctors)
        *(.rel.dtors)
    }
}

```

```

*(.rel.got)
*(.rel.dyn)
*(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
*(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
*(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
*(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
*(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
PROVIDE_HIDDEN (__rel_iplt_start = .);
*(.rel.iplt)
PROVIDE_HIDDEN (__rel_iplt_end = .);
PROVIDE_HIDDEN (__rela_iplt_start = .);
PROVIDE_HIDDEN (__rela_iplt_end = .);
}
.rela.dyn      :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
    PROVIDE_HIDDEN (__rel_iplt_start = .);
    PROVIDE_HIDDEN (__rel_iplt_end = .);
    PROVIDE_HIDDEN (__rela_iplt_start = .);
    *(.rela.iplt)
    PROVIDE_HIDDEN (__rela_iplt_end = .);
}

```

```
.rel.plt      :
{
    *(.rel.plt)
}
.rela.plt     :
{
    *(.rela.plt)
}
.init        :
{
    KEEP (*(init))
} =0
.plt         : { *(.plt) }
.iplt        : { *(.iplt) }
.text        :
{
    _ftext = . ;
    *(.text.unlikely .text.*_unlikely)
    *(.text.exit .text.exit.*)
    *(.text.startup .text.startup.*)
    *(.text.hot .text.hot.*)
    *(.text.stub .text.*.gnu.linkonce.t.*)
    /* .gnu.warning sections are handled specially by elf32.em. */
    *(.gnu.warning)
    *(.mips16.fn.*) *(.mips16.call.*)
} =0
.fini        :
{
    KEEP (*(fini))
} =0
PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
.rodata       : { *(.rodata .rodata.*.gnu.linkonce.r.*) }
.rodata1      : { *(.rodata1) }
.sdata2       :
```

```

{
    *(.sdata2 .sdata2.* .gnu.linkonce.s2.*)
}

.sbss2          : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr  : { *(.eh_frame_hdr) }
.eh_frame       : ONLY_IF_RO { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RO { *(.gcc_except_table .gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (CONSTANT (MAXPAGESIZE)) - ((CONSTANT (MAXPAGESIZE) - .) & (CONSTANT
(MAXPAGESIZE) - 1)); . = DATA_SEGMENT_ALIGN (CONSTANT (MAXPAGESIZE), CONSTANT
(COMMONPAGESIZE));
/* Exception handling */
/* Modified by ZTE-OS:
   the .rodata section is the last one defaultly arranged into data segment in  XGW media/control
process.
   We set the location counter "." to the next 32MB border,
   and the excutable segment thus occupies 32MB space in VMA*/
. = ALIGN(0x2000000);
.eh_frame       : ONLY_IF_RW { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RW { *(.gcc_except_table .gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata  : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss    : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
.preinit_array :
{
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);
}
.init_array :
{
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*(init_array))
    PROVIDE_HIDDEN (__init_array_end = .);
}

```



```
}  
.fini_array      :  
{  
    PROVIDE_HIDDEN (__fini_array_start = .);  
    KEEP (*(SORT(.fini_array.*)))  
    KEEP (*(fini_array))  
    PROVIDE_HIDDEN (__fini_array_end = .);  
}  
.ctors           :  
{  
    /* gcc uses crtbegin.o to find the start of  
       the constructors, so we make sure it is  
       first.  Because this is a wildcard, it  
       doesn't matter if the user does not  
       actually link against crtbegin.o; the  
       linker won't look for a file to match a  
       wildcard.  The wildcard also means that it  
       doesn't matter which directory crtbegin.o  
       is in.  */  
    KEEP (*crtbegin.o(.ctors))  
    KEEP (*crtbegin?.o(.ctors))  
    /* We don't want to include the .ctor section from  
       the crtend.o file until after the sorted ctors.  
       The .ctor section from the crtend file contains the  
       end of ctors marker and it must be last */  
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o) .ctors))  
    KEEP (*(SORT(.ctors.*)))  
    KEEP (*(.ctors))  
}  
.dtors           :  
{  
    KEEP (*crtbegin.o(.dtors))  
    KEEP (*crtbegin?.o(.dtors))  
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o) .dtors))  
    KEEP (*(SORT(.dtors.*)))  
    KEEP (*(.dtors))  
}
```

```
}  
.jcr          : { KEEP (*.jcr) }  
.data.rel.ro : { (*.data.rel.ro.local* .gnu.linkonce.d.rel.ro.local.*) (*.data.rel.ro* .gnu.linkonce.d.rel.ro.*) }  
. = DATA_SEGMENT_RELRO_END (0, .);  
  
.data1        : { (*.data1) }  
.got.plt      : { (*.got.plt) }  
. = .;  
_gp = ALIGN(16) + 0x7ff0;  
.got          : { (*.got) }  
/* Modified by ZTE-OS:  
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.  
   We set the location counter "." to the next 32MB border,  
   and the data segment thus occupies 32MB space in VMA*/  
.ps_spec_data : { (*.ps_spec_data)}  
.pageset      :  
{  
    __HUGE_TLB_SIZE = . ;  
    LONG(0x2000000)  
}  
.data          :  
{  
    _fdata = . ;  
    (*.data .data.* .gnu.linkonce.d.*)  
    SORT(CONSTRUCTORS)  
  
    (*.sdata .sdata.* .gnu.linkonce.s.*)  
  
    __bss_start = .;  
    _fbss = .;  
  
    (*.dynsbss)  
    (*.sbss .sbss.* .gnu.linkonce.sb.*)  
    (*.scommon)  
  
    (*.dynbss)
```

```

*(.bss .bss.* .gnu.linkonce.b.*)
*(COMMON)

/* Align here to ensure that the .bss section occupies space up to
   _end.  Align after .bss to ensure correct alignment even if the
   .bss section disappears because there are no input sections.
   FIXME: Why do we need it? When there is no .bss section, we don't
   pad the .data section.  */
. = ALIGN(. != 0 ? 64 / 8 : 1);
}

/* We want the small data sections together, so single-instruction offsets
   can access them all, and initialized data all before uninitialized, so
   we can shorten the on-disk segment size.  */

.sdata          :
{

}

.lit8            : { *(.lit8) }
.lit4            : { *(.lit4) }
.srdata          : { *(.srdata) }
_edata = .; PROVIDE (edata = .);
. = ALIGN(64 / 8);
. = ALIGN(64 / 8);
_end = .; PROVIDE (end = .);

/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 32MB border,
   and the data segment thus occupies 32MB space in VMA*/
. = ALIGN(0x2000000);

/* Modified by ZTE-OS: end */
. = DATA_SEGMENT_END (.);

/* Stabs debugging sections.  */
.stab            0 : { *(.stab) }
.stabstr         0 : { *(.stabstr) }
.stab.excl       0 : { *(.stab.excl) }
.stab.exclstr    0 : { *(.stab.exclstr) }

```

```
.stab.index      0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment        0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0.  */
/* DWARF 1 */
.debug          0 : { *(.debug) }
.line          0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo  0 : { *(.debug_srcinfo .zdebug_srcinfo) }
.debug_sfnames  0 : { *(.debug_sfnames .zdebug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges  0 : { *(.debug_aranges .zdebug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames .zdebug_pubnames) }
/* DWARF 2 */
.debug_info     0 : { *(.debug_info .gnu.linkonce.wi.* .zdebug_info) }
.debug_abbrev   0 : { *(.debug_abbrev .zdebug_abbrev) }
.debug_line     0 : { *(.debug_line .zdebug_line) }
.debug_frame    0 : { *(.debug_frame .zdebug_frame) }
.debug_str      0 : { *(.debug_str .zdebug_str) }
.debug_loc      0 : { *(.debug_loc .zdebug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo .zdebug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames .zdebug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames .zdebug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames .zdebug_typenames) }
.debug_varnames  0 : { *(.debug_varnames .zdebug_varnames) }
/* DWARF 3 */
.debug_pubtypes 0 : { *(.debug_pubtypes .zdebug_pubtypes) }
.debug_ranges   0 : { *(.debug_ranges .zdebug_ranges) }
.gnu.attributes 0 : { KEEP (*(gnu.attributes)) }
.gptab.sdata : { *(.gptab.data) *(.gptab.sdata) }
.gptab.sbss : { *(.gptab.bss) *(.gptab.sbss) }
/DISCARD/ : { *(.note.GNU-stack) *(gnu_debuglink) *(gnu_lto_*) }
}
```

64 M 对齐链接脚本，后者表示 BSS 段并入数据段。

## ■ HugeMap64.lds

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf64-tradbigmips", "elf64-tradbigmips",
              "elf64-tradlittlemips")
OUTPUT_ARCH(mips)
ENTRY(__start)
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib64");
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
nsta11/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib");
SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x120000000)); . =
SEGMENT_START("text-segment", 0x120000000) + SIZEOF_HEADERS;
    .MIPS.options : { *(.MIPS.options) }
    .note.gnu.build-id : { *(.note.gnu.build-id) }
    .dynamic       : { *(.dynamic) }
    .hash          : { *(.hash) }
    .gnu.hash      : { *(.gnu.hash) }
    .dynsym        : { *(.dynsym) }
    .dynstr        : { *(.dynstr) }
    .gnu.version   : { *(.gnu.version) }
    .gnu.version_d : { *(.gnu.version_d) }
    .gnu.version_r : { *(.gnu.version_r) }
    .rel.dyn       :
    {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro* .rel.gnu.linkonce.d.rel.ro.*)
        *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
    }
```

```

*(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
*(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
*(.rel.ctors)
*(.rel.dtors)
*(.rel.got)
*(.rel.dyn)
*(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
*(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
*(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
*(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
*(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
PROVIDE_HIDDEN (__rel_iplt_start = .);
*(.rel.iplt)
PROVIDE_HIDDEN (__rel_iplt_end = .);
PROVIDE_HIDDEN (__rela_iplt_start = .);
PROVIDE_HIDDEN (__rela_iplt_end = .);
}
.rela.dyn :
{
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
    *(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
    *(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
    *(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
    *(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
    *(.rela.ctors)
    *(.rela.dtors)
    *(.rela.got)
    *(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
    *(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
    *(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
    *(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
    *(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
    PROVIDE_HIDDEN (__rel_iplt_start = .);
    PROVIDE_HIDDEN (__rel_iplt_end = .);
}

```

```

        PROVIDE_HIDDEN (__rela_iplt_start = .);
        *(.rela.iplt)

        PROVIDE_HIDDEN (__rela_iplt_end = .);
    }

    .rel.plt      :
    {
        *(.rel.plt)
    }

    .rela.plt     :
    {
        *(.rela.plt)
    }

    .init         :
    {
        KEEP (*(init))
    } =0

    .plt          : { *(.plt) }

    .iplt         : { *(.iplt) }

    .text         :
    {
        _ftext = . ;
        *(.text.unlikely .text.*_unlikely)
        *(.text.exit .text.exit.*)
        *(.text.startup .text.startup.*)
        *(.text.hot .text.hot.*)
        *(.text.stub .text.*.gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em.  */
        *(.gnu.warning)
        *(.mips16.fn.*) *(.mips16.call.*)
    } =0

    .fini         :
    {
        KEEP (*(fini))
    } =0

    PROVIDE (__etext = .);
    PROVIDE (_etext = .);

```

```

PROVIDE (etext = .);

.rodata      : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1     : { *(.rodata1) }
.sdata2      :
{
    *(.sdata2 .sdata2.* .gnu.linkonce.s2.*)
}
.sbss2       : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
.eh_frame     : ONLY_IF_RO { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RO { *(.gcc_except_table .gcc_except_table.*) }

/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (CONSTANT (MAXPAGESIZE)) - ((CONSTANT (MAXPAGESIZE) - .) & (CONSTANT
(MAXPAGESIZE) - 1)); . = DATA_SEGMENT_ALIGN (CONSTANT (MAXPAGESIZE), CONSTANT
(COMMONPAGESIZE));

/* Exception handling  */
/* Modified by ZTE-OS:

   the .rodata section is the last one defaultly arranged into data segment in  XGW media/control
process.

   We set the location counter "." to the next 64MB border,
   and the excutable segment thus occupies 64MB space in VMA*/
. = ALIGN(0x4000000);
.eh_frame     : ONLY_IF_RW { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RW { *(.gcc_except_table .gcc_except_table.*) }

/* Thread Local Storage sections  */
.tdata  : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss   : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
.preinit_array :
{
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);
}
.init_array :
{

```



```

    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*.init_array)
    PROVIDE_HIDDEN (__init_array_end = .);
}

.fini_array      :
{
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP (*(SORT(.fini_array.*)))
    KEEP (*.fini_array)
    PROVIDE_HIDDEN (__fini_array_end = .);
}

.ctors           :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin.o(.ctors))
    KEEP (*crtbegin?.o(.ctors))

    /* We don't want to include the .ctor section from
       the crtend.o file until after the sorted ctors.
       The .ctor section from the crtend file contains the
       end of ctors marker and it must be last */
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o ) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*.ctors)
}

.dtors           :
{
    KEEP (*crtbegin.o(.dtors))

```

```

KEEP (*crtbegin?.o(.dtors))
KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o) .dtors))
KEEP (*(SORT(.dtors.*)))
KEEP (*.dtors)
}
.jcr          : { KEEP (*.jcr) }
.data.rel.ro : { (*.data.rel.ro.local* .gnu.linkonce.d.rel.ro.local.*) (*.data.rel.ro* .gnu.linkonce.d.rel.ro.*) }
. = DATA_SEGMENT_RELRO_END (0, .);
/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 64MB border,
   and the data segment thus occupies 64MB space in VMA*/
.ps_spec_data : { (*.ps_spec_data)}
.64m_pageset   :
{
    __HUGE_TLB_SIZE = . ;
    LONG(0x4000000)
}
.data          :
{
    _fdata = . ;
    (*.data .data.* .gnu.linkonce.d.*)
    SORT(CONSTRUCTORS)
}
.data1         : { (*.data1) }
.got.plt       : { (*.got.plt) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got          : { (*.got) }
/* We want the small data sections together, so single-instruction offsets
   can access them all, and initialized data all before uninitialized, so
   we can shorten the on-disk segment size.  */
.sdata        :
{
    (*.sdata .sdata.* .gnu.linkonce.s.*)
}

```

```
.lit8      : { *(.lit8) }
.lit4      : { *(.lit4) }
.srdata    : { *(.srdata) }
_edata = .; PROVIDE (edata = .);
/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 32MB border,
   and the data segment thus occupies 32MB space in VMA*/
. = ALIGN(0x4000000);
__bss_start = .;
_fbss = .;
.sbss      :
{
    *(.dynsbss)
    *(.sbss .sbss.* .gnu.linkonce.sb.*)
    *(.scommon)
}
.bss       :
{
    *(.dynbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
/* Align here to ensure that the .bss section occupies space up to
   _end.  Align after .bss to ensure correct alignment even if the
   .bss section disappears because there are no input sections.
   FIXME: Why do we need it? When there is no .bss section, we don't
   pad the .data section.  */
. = ALIGN(. != 0 ? 64 / 8 : 1);
}
. = ALIGN(64 / 8);
. = ALIGN(64 / 8);
_end = .; PROVIDE (end = .);
. = DATA_SEGMENT_END (.);
/* Stabs debugging sections.  */
.stab      0 : { *(.stab) }
.stabstr    0 : { *(.stabstr) }
```

```
.stab.excl      0 : { *(.stab.excl) }
.stab.exclstr   0 : { *(.stab.exclstr) }
.stab.index     0 : { *(.stab.index) }
.stab.indexstr  0 : { *(.stab.indexstr) }
.comment       0 : { *(.comment) }
/* DWARF debug sections.
   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0.  */
/* DWARF 1 */
.debug          0 : { *(.debug) }
.line           0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo  0 : { *(.debug_srcinfo .zdebug_srcinfo) }
.debug_sfnames  0 : { *(.debug_sfnames .zdebug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges  0 : { *(.debug_aranges .zdebug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames .zdebug_pubnames) }
/* DWARF 2 */
.debug_info     0 : { *(.debug_info .gnu.linkonce.wi.* .zdebug_info) }
.debug_abbrev   0 : { *(.debug_abbrev .zdebug_abbrev) }
.debug_line     0 : { *(.debug_line .zdebug_line) }
.debug_frame    0 : { *(.debug_frame .zdebug_frame) }
.debug_str      0 : { *(.debug_str .zdebug_str) }
.debug_loc      0 : { *(.debug_loc .zdebug_loc) }
.debug_macinfo  0 : { *(.debug_macinfo .zdebug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames .zdebug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames .zdebug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames .zdebug_typenames) }
.debug_varnames  0 : { *(.debug_varnames .zdebug_varnames) }
/* DWARF 3 */
.debug_pubtypes 0 : { *(.debug_pubtypes .zdebug_pubtypes) }
.debug_ranges   0 : { *(.debug_ranges .zdebug_ranges) }
.gnu.attributes 0 : { KEEP (*(gnu.attributes)) }
.gptab.sdata : { *(gptab.data) *(gptab.sdata) }
.gptab.sbss : { *(gptab.bss) *(gptab.sbss) }
```

```
/DISCARD/ : { *(.note.GNU-stack) *(.gnu_debuglink) *(.gnu.lto_*) }
}
```

## ■ HugeMapBss64.lds

```
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf64-tradbigmips", "elf64-tradbigmips",
              "elf64-tradlitlemips")
OUTPUT_ARCH(mips)
ENTRY(__start)
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
n stall/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib64");
SEARCH_DIR("/home/yinli/toolschain-make/V2.08.20_P2/autobuild/build/tcbuild/gcc4.5.2_glibc2.13.0/i
n stall/mips64_gcc4.5.2_glibc2.13.0/mips64-unknown-linux-gnu/lib");
SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x120000000)); . =
SEGMENT_START("text-segment", 0x120000000) + SIZEOF_HEADERS;
    .MIPS.options : { *(.MIPS.options) }
    .note.gnu.build-id : { *(.note.gnu.build-id) }
    .dynamic       : { *(.dynamic) }
    .hash          : { *(.hash) }
    .gnu.hash      : { *(.gnu.hash) }
    .dynsym        : { *(.dynsym) }
    .dynstr        : { *(.dynstr) }
    .gnu.version   : { *(.gnu.version) }
    .gnu.version_d : { *(.gnu.version_d) }
    .gnu.version_r : { *(.gnu.version_r) }
    .rel.dyn       :
    {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro* .rel.gnu.linkonce.d.rel.ro.*)
    }
```

```

*(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
*(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
*(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
*(.rel.ctors)
*(.rel.dtors)
*(.rel.got)
*(.rel.dyn)
*(.rel.sdata .rel.sdata.* .rel.gnu.linkonce.s.*)
*(.rel.sbss .rel.sbss.* .rel.gnu.linkonce.sb.*)
*(.rel.sdata2 .rel.sdata2.* .rel.gnu.linkonce.s2.*)
*(.rel.sbss2 .rel.sbss2.* .rel.gnu.linkonce.sb2.*)
*(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
PROVIDE_HIDDEN (__rel_iplt_start = .);
*(.rel.iplt)
PROVIDE_HIDDEN (__rel_iplt_end = .);
PROVIDE_HIDDEN (__rela_iplt_start = .);
PROVIDE_HIDDEN (__rela_iplt_end = .);
}
.rela.dyn :
{
*(.rela.init)
*(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
*(.rela.fini)
*(.rela.rodata .rela.rodata.* .rela.gnu.linkonce.r.*)
*(.rela.data .rela.data.* .rela.gnu.linkonce.d.*)
*(.rela.tdata .rela.tdata.* .rela.gnu.linkonce.td.*)
*(.rela.tbss .rela.tbss.* .rela.gnu.linkonce.tb.*)
*(.rela.ctors)
*(.rela.dtors)
*(.rela.got)
*(.rela.sdata .rela.sdata.* .rela.gnu.linkonce.s.*)
*(.rela.sbss .rela.sbss.* .rela.gnu.linkonce.sb.*)
*(.rela.sdata2 .rela.sdata2.* .rela.gnu.linkonce.s2.*)
*(.rela.sbss2 .rela.sbss2.* .rela.gnu.linkonce.sb2.*)
*(.rela.bss .rela.bss.* .rela.gnu.linkonce.b.*)
PROVIDE_HIDDEN (__rel_iplt_start = .);

```

```

        PROVIDE_HIDDEN (__rel_iplt_end = .);
        PROVIDE_HIDDEN (__rela_iplt_start = .);
        *(.rela.iplt)
        PROVIDE_HIDDEN (__rela_iplt_end = .);
    }
    .rel.plt      :
    {
        *(.rel.plt)
    }
    .rela.plt     :
    {
        *(.rela.plt)
    }
    .init         :
    {
        KEEP (*(init))
    } =0
    .plt          : { *(.plt) }
    .iplt         : { *(.iplt) }
    .text         :
    {
        _ftext = . ;
        *(.text.unlikely .text.*_unlikely)
        *(.text.exit .text.exit.*)
        *(.text.startup .text.startup.*)
        *(.text.hot .text.hot.*)
        *(.text.stub .text.*.gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em.  */
        *(.gnu.warning)
        *(.mips16.fn.*) *(.mips16.call.*)
    } =0
    .fini         :
    {
        KEEP (*(fini))
    } =0
    PROVIDE (__etext = .);

```

```

PROVIDE (_etext = .);
PROVIDE (etext = .);

.rodata      : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1     : { *(.rodata1) }
.sdata2      :
{
    *(.sdata2 .sdata2.* .gnu.linkonce.s2.*)
}
.sbss2       : { *(.sbss2 .sbss2.* .gnu.linkonce.sb2.*) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
.eh_frame    : ONLY_IF_RO { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RO { *(.gcc_except_table .gcc_except_table.*) }
/* Adjust the address for the data segment.  We want to adjust up to
   the same address within the page on the next page up.  */
. = ALIGN (CONSTANT (MAXPAGESIZE)) - ((CONSTANT (MAXPAGESIZE) - .) & (CONSTANT
(MAXPAGESIZE) - 1)); . = DATA_SEGMENT_ALIGN (CONSTANT (MAXPAGESIZE), CONSTANT
(COMMONPAGESIZE));
/* Exception handling */
/* Modified by ZTE-OS:
   the .rodata section is the last one defaultly arranged into data segment in  XGW media/control
process.

   We set the location counter "." to the next 64MB border,
   and the excutable segment thus occupies 64MB space in VMA*/
. = ALIGN(0x4000000);
.eh_frame    : ONLY_IF_RW { KEEP (*(eh_frame)) }
.gcc_except_table : ONLY_IF_RW { *(.gcc_except_table .gcc_except_table.*) }
/* Thread Local Storage sections */
.tdata      : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss       : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
.preinit_array :
{
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);
}
.init_array :

```



```
{
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*.init_array)
    PROVIDE_HIDDEN (__init_array_end = .);
}

.fini_array      :
{
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP (*(SORT(.fini_array.*)))
    KEEP (*.fini_array)
    PROVIDE_HIDDEN (__fini_array_end = .);
}

.ctors           :
{
    /* gcc uses crtbegin.o to find the start of
       the constructors, so we make sure it is
       first.  Because this is a wildcard, it
       doesn't matter if the user does not
       actually link against crtbegin.o; the
       linker won't look for a file to match a
       wildcard.  The wildcard also means that it
       doesn't matter which directory crtbegin.o
       is in.  */
    KEEP (*crtbegin.o(.ctors))
    KEEP (*crtbegin?.o(.ctors))

    /* We don't want to include the .ctor section from
       the crtend.o file until after the sorted ctors.
       The .ctor section from the crtend file contains the
       end of ctors marker and it must be last */
    KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o ) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*.ctors)
}

.dtors           :
{
```

```

KEEP (*crtbegin.o(.dtors))
KEEP (*crtbegin?.o(.dtors))
KEEP (*(EXCLUDE_FILE (*crtend.o *crtend?.o) .dtors))
KEEP (*(SORT(.dtors.*)))
KEEP (*.dtors)
}

.jcr          : { KEEP (*.jcr) }

.data.rel.ro : { *(.data.rel.ro.local* .gnu.linkonce.d.rel.ro.local.*) *(.data.rel.ro* .gnu.linkonce.d.rel.ro.*) }
. = DATA_SEGMENT_RELRO_END (0, .);

.data1        : { *(.data1) }
.got.plt      : { *(.got.plt) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got          : { *(.got) }
/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 64MB border,
   and the data segment thus occupies 64MB space in VMA*/
.ps_spec_data : {*(.ps_spec_data)}
.64m_pageset   :
{
    __HUGE_TLB_SIZE = . ;
    LONG(0x4000000)
}
.data          :
{
    _fdata = . ;
    *(.data .data.* .gnu.linkonce.d.*)
    SORT(CONSTRUCTORS)

    *(.sdata .sdata.* .gnu.linkonce.s.*)

    __bss_start = .;
    _fbss = .;

```

```

*(.dynsbss)
*(.sbss .sbss.* .gnu.linkonce.sb.*)
*(.scommon)

*(.dynbss)
*(.bss .bss.* .gnu.linkonce.b.*)
*(COMMON)
/* Align here to ensure that the .bss section occupies space up to
   _end.  Align after .bss to ensure correct alignment even if the
   .bss section disappears because there are no input sections.
   FIXME: Why do we need it? When there is no .bss section, we don't
   pad the .data section.  */
. = ALIGN(. != 0 ? 64 / 8 : 1);
}
/* We want the small data sections together, so single-instruction offsets
   can access them all, and initialized data all before uninitialized, so
   we can shorten the on-disk segment size.  */
.sdata      :
{

}

.lit8       : { *(.lit8) }
.lit4       : { *(.lit4) }
.srdata     : { *(.srdata) }
_edata = .; PROVIDE (edata = .);
. = ALIGN(64 / 8);
. = ALIGN(64 / 8);
_end = .; PROVIDE (end = .);
/* Modified by ZTE-OS:
   the .bss section is the first one arranged into bss/heap segment in  XGW media/control process.
   We set the location counter "." to the next 64MB border,
   and the data segment thus occupies 64MB space in VMA*/
. = ALIGN(0x4000000);
/* Modified by ZTE-OS: end */
. = DATA_SEGMENT_END (.);

```

```

/* Stabs debugging sections.  */
.stab          0 : { *(.stab) }
.stabstr       0 : { *(.stabstr) }
.stab.excl     0 : { *(.stab.excl) }
.stab.exclstr  0 : { *(.stab.exclstr) }
.stab.index    0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment       0 : { *(.comment) }

/* DWARF debug sections.

   Symbols in the DWARF debugging sections are relative to the beginning
   of the section so we begin them at 0.  */

/* DWARF 1 */
.debug         0 : { *(.debug) }
.line          0 : { *(.line) }

/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo .zdebug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames .zdebug_sfnames) }

/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges .zdebug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames .zdebug_pubnames) }

/* DWARF 2 */
.debug_info    0 : { *(.debug_info .gnu.linkonce.wi.* .zdebug_info) }
.debug_abbrev  0 : { *(.debug_abbrev .zdebug_abbrev) }
.debug_line    0 : { *(.debug_line .zdebug_line) }
.debug_frame   0 : { *(.debug_frame .zdebug_frame) }
.debug_str     0 : { *(.debug_str .zdebug_str) }
.debug_loc     0 : { *(.debug_loc .zdebug_loc) }
.debug_macinfo 0 : { *(.debug_macinfo .zdebug_macinfo) }

/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames .zdebug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames .zdebug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames .zdebug_typenames) }
.debug_varnames  0 : { *(.debug_varnames .zdebug_varnames) }

/* DWARF 3 */
.debug_pubtypes 0 : { *(.debug_pubtypes .zdebug_pubtypes) }
.debug_ranges   0 : { *(.debug_ranges .zdebug_ranges) }

```

```
.gnu.attributes 0 : { KEEP (*.gnu.attributes) }  
.gptab.sdata : { *.gptab.data *.gptab.sdata }  
.gptab.sbss : { *.gptab.bss *.gptab.sbss }  
/DISCARD/ : { *.note.GNU-stack *.gnu_debuglink *.gnu.lto_* }  
}
```

编译命令如下:

```
/home/opt/crosstools/mips/mips64_gcc4.1.2_glibc2.5.0/bin/mips64-unknown-li  
nux-gnu-gcc -static test.c -o test -Wl,-T./HugeMap32.lds
```

## ■ 限制说明

- 在同一个 XLR732 上, 两个进程使用的链接脚本类型必须一致。即不允许一个进程使用 32M 对齐链接脚本, 而另一进程使用 64M 对齐链接脚本;
  - ✧ 若第一个进程使用 32M 对齐链接脚本, 内核将启用 32M 代码数据段巨页功能;
  - ✧ 若第一个进程使用 64M 对齐链接脚本, 内核将启用 64M 代码数据段巨页功能;
- 使用“64M 代码数据段巨页功能”后, 进程的代码段和数据段属性是一致的, 因而无法单独设置属性。之前实现的“32M 代码数据段巨页功能”的代码段和数据段的属性也是设置成一样的;
- 代码数据段巨页功能的其它已知限制:
  - ✧ 不支持进程 fork(), 或者间接调用 fork() 的函数, 比如: system() 等;
  - ✧ 不能对代码段、数据段调用 munmap(), mremap() 等操作;
  - ✧ 需要 BSP 提供分配大块保留内存的接口;
  - ✧ 应用程序需要静态链接, 且最多同时支持两个进程使用此功能。

### 2.14.4. 对标准内核影响

所有代码都通过宏 CONFIG\_HUGETLB\_ELF 进行控制。当不配置该功能时, 对标准内核没有任何影响。

## 2.15. x86-64 用户态地址转物理地址接口

### 2.15.1. 修改原因

#### ■ 问题描述:

目前公司采购的内存条质量参差不齐，无论是产品线还是生产线，都要求平台提供一个有效的工具来识别缺陷内存条，为此产品线需要成研提供内核中的虚地址到物理地址的转换接口，用来获取虚地址所对应的物理地址。

#### ■ 问题分析:

标准内核没有提供物理地址到虚拟地址的转换，可以通过在 `/proc/<pid>` 下创建一个地址转换接口，实现地址转换，向该接口写入需要转换的进程虚拟地址，之后可通过 `cat` 查看虚拟地址对应的物理地址。

### 2.15.2. 修改原理和方法

- 1) 在 `/proc/pid/` 目录下添加 `vir2phy` 用来实现虚拟地址到物理地址的转换。
- 2) 在 `task_struct` 结构里增加两个字段用来保存需要查询进程的虚拟地址和物理地址。

```
unsigned long vir2phy_v;  
unsigned long vir2phy_p;
```

- 3) 根据进程虚拟地址得到页表项 `pte` 的值，再调用 `pte_pfn()` 得到页帧号。最后根据页帧号乘以页大小 `PAGE_SIZE` 同时加上偏移，即可找到虚拟地址对应物理页面的首地址。

### 2.15.3. 使用方法

- 1) 将要查询的地址写入 `vir2phy` 文件中。

```
echo va > /proc/<pid>/vir2phy
```

`va` 为需要查询的线性地址，可以为十六进制也可以为十进制

例如：

```
# echo 0x684000 > /proc/302/vir2phy
```

2) 读出虚拟地址对应的物理地址

```
cat /proc/<pid>/vir2phy
```

所得结果形式如下：

```
0xaaaaaaaa 0xbbbbbbbb
```

其中 0xaaaaaaaa 为虚拟地址，0xbbbbbbbb 为其对应的物理地址。

例如：

```
# cat /proc/302/vir2phy
0x0068400 0x07c93000
```

#### 2.15.4. 对标准内核的影响

无

## 2.16. PROC 文件获取 mips 系列 ra 寄存器的显示

### 2.16.1. 修改原因

#### ■ 问题描述：

目前产品线在定位线程异常的故障时，觉得 proc 信息不足以，还不便于分析，希望能够通过 proc 文件读取到 mips 非运行状态下线程的 ra 寄存器。为此提出希望内核做出修改，提供 proc 文件里读取 mips 非运行状态下线程的 ra 寄存器的功能。


#### ■ 问题分析：

标准内核在 proc 文件下，只对最常见的 PC、SP 寄存器的值提供了现实功能，尚没有对其他寄存器的值进行显示。可以通过在 /proc/ProcId/task/threadID 下增加一个 extra\_stat 文件实现，对内核代码做相应修改后，来返回 ra 地址。

### 2.16.2. 修改原理和方法

修改 base.c 文件中 proc\_extra\_tid\_stat 函数，在 tid 目录下增添“extra\_stat”字段用作对返回地址的显示；

在 array.c 文件中添加对 mips 的 ra 寄存器（31 号）显示处理函数，显示结果为 10 进制数。

 提示：由于测试环境限制，目前成研只有 RMI PHOENIX 单板可提供测试，故暂将该功能范围限定在 MIPS RMI PHOENIX 处理器。

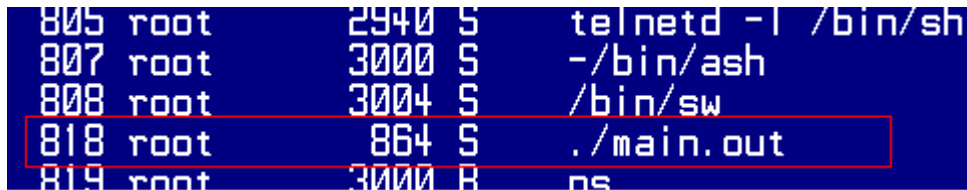
### 2.16.3. 使用方法

- 1) 确认要查看 ra 寄存器值的线程的 tid
- 2) 查看 ra 寄存器的值

```
cat /proc/Procid/task/threaded
```

示例如下：

使用 ps 命令确认要查询 ra 寄存器值的线程号如下图：



805	root	2940	S	telnetd -l /bin/sh
807	root	3000	S	-/bin/ash
808	root	3004	S	/bin/sw
818	root	864	S	./main.out
819	root	3000	R	ps

本例中进程 818 只有一个主线程号 818，故线程为 818。

通过 cat /proc/818/task/818/extra\_stat 命令查询 ra 寄存器的值，显示方式为 10 进制显示，显示结果如下：

```
# cat /proc/818/task/818/extra_stat  
7693000
```



#### 2.16.4. 对标准内核的影响

无

### 3. 开源 LICENSE 说明

#### 3.1. GPL 简介

GPL 是 GNU 通用公共许可证的简称，为大多数的 GNU 程序和超过半数的自由软件所采用，Linux 内核就是基于 GPL 协议而开源。GPL 许可协议对在 GPL 条款下发布的程序以及基于程序的衍生著作的复制与发布行为提出了保留著作权标识及无担保声明、附上完整、相对应的机器可判读源码等较为严格的要求。GPL 规定，如果将程序做为独立的、个别的著作加以发布，可以不要求提供源码。但如果作为基于源程序所生著作的一部分而发布，就要求提供源码。

成都研究所 OS 平台基于开源社区研发提供适用于各产品线的嵌入式 Linux 产品及相关工具，从总体而言，主要应该遵循 GPL 许可协议的条款。Linux 是基于 GPL2.0 发布的开源软件，CGEL 是基于 Linux 内核进行修改完成的新作品，因此，根据 GPL 协议第 2 条的规定，CGEL 属于修改后的衍生作品，并且不属于例外情况。因此，CGEL 在发布时需开放源代码，具体形式需要符合 GPL 协议第 3 条的规定。

#### 3.2. 开发指导

为尽可能地保护公司在 Linux 方面的自有研发成果，特别是产品线的应用程序，同时顺从于 GPL 协议条款的规定，这里为大家提供一些开发指导原则供参考，以便有效地规避由于顺从 GPL 条款所带来的风险。

##### ■ 应用程序尽可能运行于用户态

应用程序尽量放到用户态运行，应用程序对于内核的访问、功能使用主要通过信号、数据、文件系统、系统调用，基本属于松耦合，而且绝大部分系统调用都是通过 C 库封装进行，能比较充分地证明应用的独立性、可移植性，避免开源。

##### ■ 以动态链接方式链接开源库

以 GLIBC 库为例，应用程序对于库的链接方式应尽量采用动态链接，这样可遵循 LGPL，避免公开源码。如果需要静态链接，则可以设计一个封装容器，与开源库静态链接在一起，这样只需开放封装容

器源码。其余应用程序以动态链接方式与封装容器链接，只需要提供二进制文件。

### ■ 内核应用采用模块加载方式

内核应用可以采用静态链接、模块加载方式加入内。

静态链接方式是将所有的模块（包括应用）都编译到内核中，形成一个整体的内核映像文件。这种情况下，比较难以鉴别应用程序与内核的独立性、可移植性，按照 GPL 的规定，就很可能被要求开发源代码。

模块加载方式是指将上层应用独立地编译连接成标准 linux 所支持的模块文件 (\*.mod)。运行时，首先启动 Linux 内核，然后通过 insmod 命令将各模块按照彼此间的依赖关系插入到内核。该方式下应用程序相对独立于 Linux 内核，有可能被证明为独立的作品，与 OS 无关，认为是纯粹的聚集行为，从而可以不用开放源码。

因此，对于运行于内核态的驱动和应用软件应尽量采用模块加载方式进行独立编译，通过模块的可动态装载、卸载，以及跨 OS 的可移植性来证明应用与 OS 的独立性。

## 3.3. 源码获取方式

广东中兴新支点技术有限公司所发布的 CGEL 操作系统是基于开源软件 Linux 2.6.21 版本开发的，遵从 GPL 协议开放源代码。如果你需要源码，请发送源码申请邮件到 [cgel@gd-linux.com](mailto:cgel@gd-linux.com) 联系获取源码。