



# D1-H Tina Linux 内存优化 开发指南

**版本号: 1.0**

**发布日期: 2021.04.07**

## 版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.07	AWA0916	first version



# 目 录

<b>1 概述</b>	<b>1</b>
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
<b>2 内存使用情况分析</b>	<b>2</b>
2.1 DRAM 大小	2
2.2 系统内存使用情况	3
2.2.1 free 命令	3
2.2.2 /proc/meminfo 节点	4
2.3 保留内存	5
2.4 buffers & cached	7
2.5 系统使用的内存	8
2.5.1 进程使用的内存	8
2.5.2 总使用内存	8
<b>3 内存优化</b>	<b>10</b>
3.1 保留内存优化	10
3.1.1 内核静态内存优化	10
3.1.2 DTB 内存优化	10
3.1.3 opensbi 预留内存优化	11
3.1.4 disp 预留内存优化	11
3.2 内核使用内存优化	11
3.3 Slab 优化	11
3.4 内核模块优化	11
3.5 用户空间使用内存优化	11

# 1 概述

## 1.1 编写目的

介绍 Tina Linux 下减少系统使用内存的方法。

## 1.2 适用范围

适用于硬件平台：全志 D1-H 芯片。

软件平台：Tina v3.5 及后续版本。

## 1.3 相关人员

适用于 TinaLinux 平台的客户及相关技术人员。

## 2 内存使用情况分析

内存优化通常分为三个阶段：

- 明确目标

优化无止境，优化程度越大，优化难度与工作量就越大，代码也会变得越不通用。明确优化的目标非常重要。

- 了解现状

了解当前系统的总内存，剩余内存，各部分内存占用情况等。

- 评估优化

对内存使用现状进行评估，针对性的进行优化。

本章说明系统当前内存的使用情况。

### 2.1 DRAM 大小

硬件上 DDR 确定之后，DRAM 大小就已经确定。

boot0/uboot 会根据 DRAM 驱动提供的接口获取 DRAM 的大小，然后修改 dts 中的 memory 节点，Linux 启动时解析 dts 获取 DRAM 的大小。

启动 log 中会打印 dram 的大小。比如 D1-H 方案 boot0/uboot/kernel 启动时会有如下 log：

```
[276]DRAM SIZE =512 M
...
[00.705]DRAM:  512 MiB
...
[  0.000000] Memory: 491608K/522240K available
```

执行 `hexdump -C /sys/firmware/devicetree/base/memory@40000000/reg` 也可以获取 dram 的起始地址与大小。如下面例子所示，其中 0x40000000 为起始地址，0x20000000 为 dram 的 size。

```
root@TinaLinux:/# hexdump -C /sys/firmware/devicetree/base/memory@40000000/reg
00000000  00 00 00 00 40 00 00 00 00 00 00 00 20 00 00 00  |....@..... ...|
00000010
```

## 2.2 系统内存使用情况

### 2.2.1 free 命令

进入 Linux 用户空间，执行 free 命令可获得当前系统的内存使用情况。

比如 D1-H 方案，某次执行 free 命令的结果如下：

```
root@TinaLinux:/# free
              total        used        free      shared    buffers     cached
Mem:          499872      43508      456364         36       3940       8068
-/+ buffers/cache:      31500      468372
Swap:           0           0           0
```

free 命令输出说明如下：

- 第一行 Mem，当前系统内存的使用情况。

- total：Linux 内核可支配的内存。
- used：系统已使用的内存。
- free：系统尚未使用的内存。
- shared：共享内存以及 tmpfs、devtmpfs 所占用的内存。共享内存指使用 shmget、shm\_open、mmap 等接口创建的共享内存。
- buffers：Buffers 表示块设备 block device 所占用的缓存页，包括直接读写块设备、以及文件系统元数据 metadata 等。
- cached：Cache 里包括所有与文件对应的内存页。如果一个文件不再与进程关联，该文件不会立即回收，此时仍然包含在 Cached 中；此外，Cached 中还包含 tmpfs 中的文件以及 shmem 等。

- 第二行 -/+ buffers/cache，减号表示第一行 used 内存减去 buffers 与 cached 内存，即  $\text{Mem\_used} - \text{Mem\_buffers} - \text{Mem\_cached}$ ；加号表示第一行 free 内存加上 buffers 与 cached 内存，即  $\text{Mem\_free} + \text{Mem\_buffers} + \text{Mem\_cached}$ 。从应用程序的角度看，buffers 和 cached 是潜在可用的内存。
- 第三行：Swap，交换分区的使用情况。Tina 产品上使用 flash 作为存储器，读写次数是有限的，而 swap 分区特点是会被频繁地读写，导致 flash 寿命变短，因此 tina 上没有创建 swap 分区。

- total: 交换分区总大小。
- used: 已使用的交换分区大小。
- free: 空闲的交换分区大小。

## 2.2.2 /proc/meminfo 节点

实际上 free 命令信息来源是从/proc/meminfo 节点。

比如 D1-H 方案，某次执行cat /proc/meminfo命令的结果如下：

```
root@TinaLinux:/# cat /proc/meminfo
MemTotal:      499872 kB
MemFree:       456332 kB
MemAvailable:  463236 kB
Buffers:       3940 kB
Cached:        8068 kB
SwapCached:    0 kB
Active:        9892 kB
Inactive:      4360 kB
Active(anon):  2256 kB
Inactive(anon): 24 kB
Active(file):  7636 kB
Inactive(file): 4336 kB
Unevictable:   0 kB
Mlocked:       0 kB
SwapTotal:     0 kB
SwapFree:      0 kB
Dirty:         0 kB
Writeback:     0 kB
AnonPages:     2292 kB
Mapped:        3384 kB
Shmem:         36 kB
KReclaimable:  5380 kB
Slab:          12948 kB
SReclaimable:  5380 kB
SUnreclaim:    7568 kB
KernelStack:   1040 kB
PageTables:    288 kB
NFS_Unstable:  0 kB
Bounce:        0 kB
WritebackTmp:  0 kB
CommitLimit:   249936 kB
Committed_AS:  63148 kB
VmallocTotal:  67108863 kB
VmallocUsed:    4284 kB
VmallocChunk:   0 kB
Percpu:        32 kB
CmaTotal:      8192 kB
CmaFree:       6212 kB
```

相关说明如下：

- MemTotal、MemFree、Buffers、Cached、Shmem，即 free 命令第一行。

- MemAvailable：使用 MemFree, Active(file), Inactive(file), SReclaimable 和/proc/zoneinfo 中的 low watermark 根据特定算法计算得出，是一个估值，并不精确。可用来评估应用程序层面可用的内存。
- Active/Inactive：Active 表示最近使用的内存，回收的优先级低；Inactive 表示最近较少使用的内存，回收的优先级高。可细分为 Active/Inactive 匿名页与文件页。所谓文件页，就是与文件对应的内存页，如进程的代码、映射的文件都属于文件页，当内存不足时，这部分的内存可以写回到存储器中；与之对应的就属于匿名页，即没有与具体文件对应的页，如进程的堆栈等，内存不足时，如果存在 swap 分区，可以将匿名页写入到交换分区，如果没有 swap 分区，则只能常驻内存中。
- AnonPages：匿名页。
- Mapped：设备或文件映射的大小。比如共享内存、动态库、mmap 的文件等都统计在该内存中。
- slab/SReclaimable/SUnreclaim：内核 slab 使用的内存，包含可回收与不可回收部分。
- KernelStack：内核栈大小
- PageTables：页表的大小（用于将虚拟地址翻译为物理地址），内存分配越多，此块内存就会增大。
- CommitLimit/Committed\_AS：overcommit 的阈值/已经申请的内存大小（不是已分配）。Overcommit 是 Linux 一种内存申请处理方式，为了跑更多更大的程序，大部分申请内存的请求都回复“yes”，总申请的内存大于总物理内存。
- VmallocTotal/VmallocUsed/VmallocChunk：vmalloc 区域大小/vmalloc 区域使用大小/vmalloc 区域中最大可用的连续区块大小。
- CmaTotal/CmaFree：总 CMA 内存/空闲 CMA 内存。

## 2.3 保留内存

D1-H DRAM 大小为 512M，而 free 命令中显示系统可支配总内存只有 499872KB=488M，这里就涉及到一个概念：保留内存（Reserved Memory）。

保留内存是指把系统中的一部分内存保留起来用作特殊用途，这部分内存通常不会被释放，也不会被转移到交换分区。

进入控制台，执行 `cat /sys/kernel/debug/memblock/reserved` 可以查看 reserved memory 使用情况。

当前 D1-H reserved memory 使用情况如下：

```
root@TinaLinux:/# cat /sys/kernel/debug/memblock/reserved
0: 0x0000000040000000..0x0000000040b1839b
1: 0x0000000041200000..0x000000004120000b
2: 0x0000000042000000..0x00000000420fffff
3: 0x000000005e500000..0x000000005f3fffff
4: 0x000000005f5af580..0x000000005f93957f
```



```
5: 0x000000005fef9bc0..0x000000005ffb9fc7
6: 0x000000005ffba000..0x000000005ffc1fff
7: 0x000000005ffc3100..0x000000005ffc3678
8: 0x000000005ffc3680..0x000000005ffc376f
9: 0x000000005ffc3780..0x000000005ffc3787
10: 0x000000005ffc37c0..0x000000005ffc37c3
11: 0x000000005ffc3800..0x000000005ffc3807
12: 0x000000005ffc3840..0x000000005ffc3e47
13: 0x000000005ffc3e78..0x000000005ffc3ea3
14: 0x000000005ffc3ea8..0x000000005ffc3ed3
15: 0x000000005ffc3ed8..0x000000005ffc3f03
16: 0x000000005ffc3f08..0x000000005ffc3f32
17: 0x000000005ffc3f38..0x000000005ffc3f62
18: 0x000000005ffc3f68..0x000000005ffc3f93
19: 0x000000005ffc3f98..0x000000005ffc3fc3
20: 0x000000005ffc3fc8..0x000000005ffc3ff3
21: 0x000000005ffc3ff8..0x000000005ffc4023
22: 0x000000005ffc4028..0x000000005ffc4053
23: 0x000000005ffc4058..0x000000005ffc4083
24: 0x000000005ffc4088..0x000000005ffc40b3
25: 0x000000005ffc40b8..0x000000005ffc40e3
26: 0x000000005ffc40e8..0x000000005ffc4113
27: 0x000000005ffc4118..0x000000005ffc4143
28: 0x000000005ffc4148..0x000000005ffc4173
29: 0x000000005ffc4178..0x000000005ffc41a3
30: 0x000000005ffc41a8..0x000000005ffc41d3
31: 0x000000005ffc41d8..0x000000005ffc4203
32: 0x000000005ffc4208..0x000000005ffc4233
33: 0x000000005ffc4238..0x000000005ffc4266
34: 0x000000005ffc4268..0x000000005ffc4296
35: 0x000000005ffc4298..0x000000005ffc42c6
36: 0x000000005ffc42c8..0x000000005ffc42f6
37: 0x000000005ffc42f8..0x000000005ffc4326
38: 0x000000005ffc4328..0x000000005ffc4356
39: 0x000000005ffc4358..0x000000005fffef9f
40: 0x000000005fffefc0..0x000000005fffffff
```

在内核 cmdline 加上 memblock=debug bootmem\_debug=1 参数，在内核启动时，会打印上述 reserved memory 详细信息。由于内容太多，这里不展示了。

经分析对比，当前 D1-H reserved memory 主要包含如下几个部分：

- 0: 0x0000000040000000..0x0000000040b1839b, size:11360K

opensbi 预留 2M, 0x0000000040000000-0x0000000040200000

其他 0x0000000040200000-0x0000000040b1839b 为内核代码段、数据段。其中内核包括 text, init, data, bss 四段，其中 init 在内核启动完成后会被释放。

- 2: 0x0000000042000000..0x00000000420fffff, size:1024K

DSP 内存（共 1M）。

- 3: 0x000000005e500000..0x000000005f3fffff, size:15360K

所有 struct page 结构体的总大小, 0x000000005e500000-0x000000005ebfffff, 共 7M。struct page 结构体用来描述物理上的页帧, 当前 D1-H 上配置一个页的大小为 4K。

CMA 内存 0x000000005ec00000-0x000000005f3fffff, 共 8M, 在 cmdline 中通过 cma=8M 配置而来。在初始化的过程中, CMA 内存会全部导入伙伴系统, 所以内核是可以支配 CMA 内存的。

- 4: 0x000000005f5af580..0x000000005f93957f, size:3624K

disp 保留内存。

- 5: 0x000000005fef9bc0..0x000000005ffb9fc7, size:769K

主要是 vfs cache, 包括 Dentry 和 Inode 的 hash table, 存放最近访问的 Dentry 和 Inode 节点, 加速对虚拟文件系统的访问。

- 39: 0x000000005ffc4358..0x000000005ffef9f, size:235K

主要是解析 dtb 生成 struct device\_node 结构体所用的内存。

## 2.4 buffers & cached

free 命令第二行 -/+ buffers/cache, 隐含的意思是 buffers 与 cached 内存都属于空闲内存, 实际上并非如此。

Linux 为加速 IO 访问速度, 会使用空闲内存来缓存文件以及元数据等内容, 这就是 buffers 和 cached 内存。当内存不足时, 这些内存会被回收, 供内核与应用使用。

所以 buffer 与 cache 实际上是已经使用了的内存, 由于可以回收, 属于潜在的空闲内存。

但是并非所有的 buffer 和 cache 都可以回收, 比如:

- 如果有某个进程访问块设备或者普通文件, 就需要 buffers 和 cached 空间, 这部分就不能释放。
- shared、tmpfs 也包含在 cached 空间中。

## 2.5 系统使用的内存

### 2.5.1 进程使用的内存

新增一个进程使用了哪些内存？

首先，访问文件系统加载进程的可执行文件、库等，导致 buffer/cache 增大；其次，进程本身在用户空间运行时需要有自己的地址空间信息（用 mm\_struct 结构体来表示，包含代码段、数据段、用户栈等地址空间描述）；再次，内核会为进程创建进程描述符（task\_struct）、内存描述符（mm\_struct）等结构体，用于管理进程；此外，进程还有对应的内核栈，当进程陷入内核时需要内核栈来支持内核函数调用等等。

我们常说的进程使用的内存，指的是在用户空间所使用的内存。

关于用户进程的内存使用，涉及几个通用概念：

- VSS: Virtual Set Size 使用的虚拟内存（包含共享库占用的内存）
- RSS: Resident Set Size 实际使用物理内存（包含共享库占用的内存）
- PSS: Proportional Set Size 实际使用的物理内存（比例分配共享库占用的内存）
- USS: Unique Set Size 进程独自占用的物理内存（不包含共享库占用的内存）

一般来说：VSS >= RSS >= PSS >= USS

在 /proc/<PID>/smaps 节点中包含了进程的每一个内存映射的统计值，包含了 PSS、RSS 等信息。所以对 /proc/<PID>/smaps 节点中所有的 PSS 进行累加，即可统计出所有进程在用户空间所使用的内存，具体命令如下：

```
grep ^Pss /proc/[0-9]*/smaps | awk '{total+=$2}; END {print total}'
```

### 2.5.2 总使用内存

单一个进程，涉及到了很多种类的内存使用，完全统计起来不太现实。为统计系统总使用内存，可将其划分为用户空间使用内存与内核使用内存。

用户空间使用的内存 = Buffers + Cached + AnonPages。

内核使用的内存 = Slab + PageTable + KernelStack + CmaUsed + Vmalloc + X。

其中，

- CmaUsed = CmaTotal - CmaFree。

- Vmalloc 表示/proc/vmallocinfo 中的 vmalloc 分配的内存，包含了内核模块使用的内存。计算方法为 `grep vmalloc /proc/vmallocinfo | awk '{total+=$2}; END {print total}'`。
- X 表示直接通过 `alloc_pages/get_free_page` 分配的内存，这部分内存未纳入统计，属于内存黑洞。



## 3 内存优化

本章将介绍一些通用的优化方法，主要包括：

- 保留内存优化。
- 内核使用内存优化。
- 用户空间使用内存优化。

### 3.1 保留内存优化

#### 3.1.1 内核静态内存优化

内核静态内存包括内核代码段数据段。优化方法主要有如下几种：

- 1) 关闭不需要的模块，关闭模块下不需要的功能。

在内核根目录，执行 `scripts/ksize vmlinux` 各个模块的代码段数据段的统计信息。

- 2) 关闭内核调试功能。
- 3) 开启 `CONFIG_CC_OPTIMIZE_FOR_SIZE` 宏，使能 `-Os` 编译参数。
- 4) 排查内核占用空间大的符号。

执行 `nm --size -r vmlinux`，可以列出所有符号占用的内存。

#### 3.1.2 DTB 内存优化

Tina 内核中提供的 DTS 一般来说比较全面，对于特定的方案，往往用不了那么多，可以针对性的删除一些节点。

### 3.1.3 opensbi 预留内存优化

本例中 opensbi 预留内存过大，可咨询 AW 内部接口负责人进行优化。

### 3.1.4 disp 预留内存优化

本例中 disp 预留内存过大，如果不启用 uboot show logo 可省掉这块内存，具体可咨询 AW 内部接口负责人进行优化。

## 3.2 内核使用内存优化

由 2.5.2 小节可知，内核使用的内存包括 Slab、PageTable、KernelStack、CmaUsed、Vmalloc 等。

### 3.3 Slab 优化

目前 Tina 上大部分方案默认选用的是 SLUB 分配器。

- 1) 关闭 slab 调试宏 COFNIG\_SLUB\_DEBUG 与 CONFIG\_SLABINFO。
- 2) 尝试使用针对微小的嵌入式系统的 SLOB。

### 3.4 内核模块优化

- 1) 不要开机全部加载，实时加载，实时卸载。
- 2) 将内核模块编译到内核镜像中。

### 3.5 用户空间使用内存优化

- 1) 使用更小的 C 库。
- 2) 使用 size 优化的编译选项，比如 -Os, -mthumb 等。
- 3) 将 tmpfs 下大文件保持到 flash 上。
- 4) 使用更小的库或应用程序。比如使用 mbedtls，而不是 openssl。
- 5) 减少守护进程数量，实时运行/关闭特定程序。

- 6) 将只被一次依赖的动态库转化为静态库。
- 7) 使用 `dlopen` 来控制动态库的生存周期。
- 8) 优化程序源码。



## 著作权声明

版权所有 © 2022 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。