Tina Linux 内存优化
开发指南

版本号: 1.1
发布日期: 2021.04.20

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.07.22	AWA0916	first version
1.1	2021.04.20	AWA0916	新增硬件平台支持。

目　录

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

硬件平台：全志 R/V/F/MR 系列芯片。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。

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。

uboot 启动 log 中会打印 dram 的大小。比如 R329 方案 uboot 启动时会有如下 log：

```
[01.300]DRAM:  128 MiB
```

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 R329 例子所示，其中 0x40000000 为起始地址，0x08000000 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 08 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 R329 方案，某次执行 free 命令的结果如下：

```
root@TinaLinux:/# free
             total       used       free     shared    buffers     cached
Mem:        110592      79960      30632         36       9172      22860
-/+ buffers/cache:      47928      62664
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`Mem_used - Mem_buffers - Mem_cached`；加号表示第一行 free 内存加上 buffers 与 cached 内存，即`Mem_free + Mem_buffers + 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 R329 方案，某次执行`cat /proc/meminfo`命令的结果如下：

```
root@TinaLinux:/# cat /proc/meminfo
MemTotal:         110592 kB
MemFree:           30380 kB
MemAvailable:      62044 kB
Buffers:            9276 kB
Cached:            22872 kB
SwapCached:            0 kB
Active:            17580 kB
Inactive:          16336 kB
Active(anon):       1780 kB
Inactive(anon):       24 kB
Active(file):      15800 kB
Inactive(file):    16312 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:          1800 kB
Mapped:             3324 kB
Shmem:                36 kB
Slab:              13536 kB
SReclaimable:       4224 kB
SUnreclaim:         9312 kB
KernelStack:        1232 kB
PageTables:          200 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:       55296 kB
Committed_AS:      29116 kB
VmallocTotal:   263061440 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
CmaTotal:          24576 kB
CmaFree:            2684 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。这部分在新版本内核上移除了（详见 https://github.com/torvalds/linux/commit/a5ad88ce8c7fae7ddc72ee49a11a75aa837788e0）。
- CmaTotal/CmaFree：总 CMA 内存/空闲 CMA 内存。

2.3 保留内存

R329 DRAM 大小为 128M，而 free 命令中显示系统可支配总内存只有 110592KB=108M，这些看不到的内存属于保留内存（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 R329 reserved memory 使用情况如下：

```
root@TinaLinux:/# cat /sys/kernel/debug/memblock/reserved
   0: 0x0000000040080000..0x00000000408a3fff, size:8336K
   1: 0x00000000408a5000..0x00000000408a5fff, size:4K
   2: 0x0000000041700000..0x000000004171767f, size:93K
   3: 0x0000000041800000..0x00000000420fffff, size:9216K
```

```
  4: 0x0000000045000000..0x00000000467fffff, size:24576K
  5: 0x0000000046d86000..0x0000000046f85fff, size:2048K
  6: 0x0000000047f29e00..0x0000000047f59e5f, size:192K
  7: 0x0000000047f59e80..0x0000000047f59edf, size:0K
  8: 0x0000000047f59f00..0x0000000047f8400f, size:168K
  9: 0x0000000047f84080..0x0000000047f84087, size:0K
 10: 0x0000000047f84100..0x0000000047f84107, size:0K
 11: 0x0000000047f85180..0x0000000047fff2e6, size:488K
 ......
```

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 R329 reserved memory 主要包含如下几个部分：

- 0: 0x0000000040080000..0x00000000408a3fff, size:8336K

内核代码段、数据段。详细包括 text，init，data，bss 四段，其中 init 在内核启动完成后会被释放。

- 1: 0x00000000408a5000..0x00000000408a5fff, size:4K

略。

- 2: 0x0000000041700000..0x000000004171767f, size:93K

DTB 占用内存。

- 3: 0x0000000041800000..0x00000000420fffff, size:9216K

TEE 内存（共 8M，包括 SHM 2M，ATF 1M，OS 1M，TA 4M）。

DSP 内存（共 1M）。

- 4: 0x0000000045000000..0x00000000467fffff, size:24576K

CMA 内存，共 24M，在 cmdline 中通过`cma=24M`配置而来。在初始化的过程中，CMA 内存会全部导入伙伴系统（具体是通过 cma_init_reserved_areas 函数来实现），所以内核是可以支配 CMA 内存的。

- 5: 0x0000000046d86000..0x0000000046f85fff, size:2048K

所有 struct page 结构体的总大小。struct page 结构体用来描述物理上的页帧。当前 R329 上配置一个页的大小为 4K，因此总共有 128M / 4K = 32768 个页，而 sizeof(struct page) 的值为 64B，因此这一块共 32768 * 64 = 2048 KB。注：aarch64 内核 sizeof(struct page) 的值为 64B，arm32 内核 sizeof(struct page) 的值为 32B。

- 6: 0x0000000047f29e00..0x0000000047f59e5f, size:192K

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。

- 8: 0x0000000047f59f00..0x0000000047f8400f, size:168K

主要是 per-cpu 变量占用的内存。

- 11: 0x0000000047f85180..0x0000000047fff2e6, size:488K

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.py 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。

5）开启 CONFIG_THUMB2_AVOID_R_ARM_THM_JUMP11 和 CONFIG_THUMB2_KERNEL。
注：会带来性能损失，具体损失根据实际平台典型应用场景进行测试。

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 TEE 内存优化

Tina 中一些平台，会默认配置一些 TEE 保留内存。对于特定方案来说，很有可能用不了那么多内存。TEE 默认保留配置与`bl31.bin`、`optee-${CHIP}.bin`是配套的，因此修改时需要注意同步更新。

比如 R329 默认配置了 8M 内存（SHM 2M,ATF 1M,OS 1M,TA 4M），但是对于非安全方案来说，只需要保留 ATF 就够了；对于不使用 TA 的安全方案，只需要保留 ATF 与 OS 的内存就可以了。

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) 对于 64 位 CPU，可以使用 32 位的 rootfs。
5) 使用更小的库或应用程序。比如使用 mbedtls，而不是 openssl。

6) 减少守护进程数量，实时运行/关闭特定程序。

7) 将只被一次依赖的动态库转化为静态库。

8) 使用 dlopen 来控制动态库的生存周期。

9) 优化程序源码。

著作权声明

商标声明

免责声明