信息科学与技术丛书 移动与嵌入式开发系列

史子旺　叶超群　蔡建宇　编著

嵌入式Linux内存使用与性能优化

- 让程序占用内存更少
- 让系统启动速度更快
- 逻辑优化与代码优化的辩证关系
- 系统性能优化

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书·移动与嵌入式开发系列

嵌入式 Linux 内存使用与性能优化

史子旺　叶超群　蔡建宇　编著

机械工业出版社

本书主要讲述嵌入式系统开发中的两个难点：系统的内存使用与系统性能优化。

关于系统的内存使用，本书试图完成两项任务：① 增加系统整体的空闲内存数量，从而提高进程的并发能力；② 使系统在长时间运行后，仍然能够保持较高数量的空闲内存。本书所讲述的内存使用，不是 Linux 内核的内存管理，而是在用户层观察进程是如何使用内存的。

关于系统性能优化，本书不同于同类书侧重编程语法或者发挥硬件性能的做法，而是着眼于大型软件项目性能优化实践，阐明了逻辑优化与代码优化的辩证关系，提出了软件优化层次的概念。针对代码优化，阐述了代码优化的境界，为代码优化指明了研究方向。对于编程过程中的一些常用元素，通过查看汇编代码的方式，阐述了其性能上的差别。

本书适合嵌入式系统开发人员阅读，也可供有一定经验的 C/C++程序员和 Linux 程序员参考。

图书在版编目（CIP）数据

嵌入式 Linux 内存使用与性能优化 / 史子旺，叶超群，蔡建宇编著. —北京：机械工业出版社，2009.5

（信息科学与技术丛书 • 移动与嵌入式开发系列）

ISBN 978-7-111-27043-0

Ⅰ. 嵌… Ⅱ. ①史…②叶…③蔡… Ⅲ. Linux 操作系统－程序设计 Ⅳ. TP316.89

中国版本图书馆 CIP 数据核字（2009）第 070659 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：车 忱

责任编辑：车 忱

责任印制：乔 宇

北京双青印刷厂印刷

2009 年 5 月第 1 版 • 第 1 次印刷

184mm × 260mm • 20.5 印张 • 502 千字

0 001 — 3 500 册

标准书号：ISBN 978-7-111-27043-0

定价：39.00 元

凡购本图书，如有缺页、倒页、脱页，由本社发行部调换

销售服务热线电话：（010）68326294 68993821

购书热线电话（010）88379639 88379641 88379643

编辑热线电话（010）88379753 88379739

封面无防伪标均为盗版

出版说明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术和新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术不断地受到挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着Internet延伸到人类世界的方方面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用。所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络和工程应用等内容，注重理论与实践的结合，内容实用、层次分明、语言流畅。是信息科学与技术领域专业人员不可或缺的参考书。

目前，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前　言

书名为《嵌入式 Linux 内存使用与性能优化》，说实话笔者有些心虚。嵌入式设备的种类太多，各种用法不尽相同，关于 Linux 操作系统的剪裁方面的文章就更多了，笔者不可能都十分了解，只了解自身正在使用的系统——智能手机和 Linux 2.6（MontaVista 3.4.3）。但笔者坚信这本书中有些东西还是通用的，希望本书能对读者的工作有所帮助。也正是因为笔者在一家手机公司工作，所以本书中多数内容的讲解以手机为例。

上学的时候，老师就告诉我们，“不要关注内存，现在内存很便宜；不要过多强调编码技巧，现在 CPU 速度已经很快了，我们应该更加看重代码的结构和良好的扩展性、可读性。”正是这句话，使得很多程序员找到了其代码效率低下的借口。对于 PC 来讲，这句话也许是对的，但对于嵌入式设备来讲，则不得不把代码的效率提到一个很高的位置，因为我们听到了用户太多的抱怨：手机为什么这么慢！

在这里，首先欢迎你加入到嵌入式 Linux 软件开发的行列，你处于一个最有前途的行业，全球的著名 IT 公司都在这里竞争，而其还处在高速发展阶段，可以为你提供广阔的发展空间。

其次要告诉你几个坏消息，希望你有心理准备：

（1）目前还是“战国时代”，公司与公司之间，平台与平台之间竞争激烈，这对开发人员而言就意味着更多的加班。

（2）开发环境异常恶劣，现在大部分智能设备的 CPU 主频还处在 200～500MHz 之间，与笔者 2000 年买的奔腾 II 计算机处于一个水平；内存 32～64MB，在当前 1GB 内存也就 100 多元的时代，公司可谓吝啬之极。更糟糕的是，在手机上没有高性能的显卡，这就意味着 CPU 要承担更多的图形处理功能。因此，在嵌入式系统中，CPU 的负荷是很高的，而用户对软件性能的要求并没有减弱，这就对程序的性能提出了更高的要求。

（3）调试手段单一，目前软件的调试还处于在代码中增加一些打印日志的状态，虽然也有 GDB，也有图形化的 IDE 开发环境，但并没有广泛使用，这对于 UNIX/Linux 程序员来讲是很自然的，但对于从 Windows 转移过来的程序员来讲，则显得十分丑陋。

不过这不应该由我来抱怨，正因如此才有我的存在，才有我们作为系统优化人员存在的价值，才能拿着高薪。

在书店里，充斥着关于 Linux 的书籍，教你如何编写程序，解读 Linux 内核，但是很少讲述程序如何运行、glibc 如何进行内存管理、程序与系统如何交互等，而这正是做系统优化所不得不关心的问题。

这些知识涉及面很广，分布在不同的地方，需要开发人员不断积累、整理。从事软件优化工作快两年了，笔者尝试着把工作过程中所学习到的知识进行总结，不清楚的地方去研究，争取做到知识体系系统化，这样，一方面提高自己的水平，另一方面，也希望对大家的工作有些帮助。

笔者的博客是 http://blog.chinaunix.net/u/30686/，主要收集和发表关于嵌入式 Linux 软件性能优化方面的文章。笔者的 E-mail 是 loughsky@sina.com，如果您有什么问题，可以给我发

送邮件，我将尽全力解答。

本书的结构

本书主要解决在嵌入式 Linux 软件开发过程中所遇到的两大难题：进程的内存使用和软件的性能优化。

第一部分主要介绍系统的内存优化，所要完成的任务有两个：

- 减少每个进程的内存使用，增加系统空闲内存数量。

第 1 章讲述如何评估一个进程的内存使用。

第 2 章详细讲述进程的内存使用。

第 3 章讲述如何从系统级别调整内存使用量。

- 发现和定位内存泄漏。

第 4 章详细讲述了如何发现内存泄漏，并讲述了将内存泄漏定位到代码行的具体方法。

第二部分主要介绍软件性能优化，怎样使软件跑得飞快。

第 5 章讲述性能优化的工作流程。

第 6 章讲述进程的启动速度，这是软件性能中的一个很重要的指标，由于其优化方法不同，故单独列为一章。

第 7 章讲述性能优化的方法。俗话说授人以鱼不如授人以渔，本章更加注重软件优化过程中的方法和理论，非常重要。

第 8 章讲述代码优化的境界，主要讲述一些编码技巧，涉及编译器和 ARM 芯片的一些特性等。

第 9 章讲述系统性能优化，前面 4 章侧重于单个进程自身的性能优化，而系统所表现的性能往往是多个进程性能相互作用的结果。本章将侧重系统性能的调整。

本书主要面向的读者：

- 嵌入式 Linux 软件开发人员。
- 系统优化人员。
- 系统架构师。

致谢

本书由史子旺、叶超群、蔡建宇编写，其中叶超群、蔡建宇负责第一部分内存使用的编写工作，史子旺完成了第二部分软件性能优化。本书的写作占用了作者大量的和家人在一起的时间，在此感谢作者家人们的理解和支持。

史子旺

目　录

第一篇 内存使用

记得那是前年的一个冬天，笔者所在团队成员都很高兴，因为新设备上要增加的功能已经完成得差不多了，可以在上班时喝喝茶、聊聊天，幻想着周末去滑雪。这时老板把我们召集到一起开了个会。

走进会议室，老板在那里摆弄着手机，同时招手示意我们坐下。

我们坐在那里，静静地、疑惑地看着老板。

“好了。”老板从座位上站了起来，把手机递给了我们。

我们一看，手机上显示“Memory Low”（内存不足），再看看手机上运行着地址本、Java小游戏，还有MP3。

“这不是我们的问题，哪个手机也不能同时运行这么多进程。”

“用户不该这样使用，他应该退出Java游戏，然后再去听歌。”

……

老板摆了摆手，“不要说这些，我们不能告诉用户如何使用手机。内存，现在最重要的问题是把系统使用内存减下去，每个进程都要减20%。”

我们知道老板是对的，这样的东西是无法交付给用户的。

会后每个小组都去检查自己的进程内存使用情况，而笔者的任务则是如何推动内存的优化。

当接受到一项任务时，笔者向来是采用这样的方法：

第一、明确目标。做一件事情之前，必须要知道干什么，否则就不知道往哪个方向去努力。

第二、了解当前的状态。在确定了目标之后，才能确定需要了解的内容，同时试图找到方法来评估当前的状态。在找到了评估方法之后，才能了解当前水平与目标的差距，做到心里有数，从而做到知己知彼，百战百胜。也正是有了评估方法，才能不断地尝试各种优化方法，验证其优化的效果。

第三、尝试各种优化方法，验证优化效果。

如果不能满足目标，需要返回到第二步。

软件的优化，往往不是一次优化所能够实现的，它是多个优化累积的结果，优化本身也是一个不断迭代的过程。

针对系统内存的优化，笔者仍然遵从这个过程。

第一、明确目标。针对于系统内存的优化，主要有两个目标：

（1）每个守护进程使用的内存要尽可能少，这样系统的剩余内存才会多，才能够支撑更多的应用同时在系统内运行。

（2）系统在长时间运行之后，各个守护进程仍然保持着较低的内存使用。这就要求进程在运行过程中没有内存泄漏，在长时间运行之后，系统仍能保持着一个较高的剩余内存。

第二、了解当前的状态。需要找到一种方法，来评估系统的空闲内存和各个进程的内存使用情况。

第1章将详细介绍内存的评估方法。

第三、尝试各种优化方法。针对目标 1，第 2 章将详细介绍执行文件各个部分的内存使用情况，同时也给出了优化方法。

第 3 章将详细介绍 Linux 系统本身内存的使用情况。

针对目标 2，第 4 章将详细介绍如何检测、发现和修正内存泄漏。

在书中，涉及在嵌入式设备和交叉编译环境中运行的一些命令，这些命令将全部使用**黑体**显示。

在嵌入式环境下运行的命令，将以#开头，例如 **#busybox free**。

在交叉编译环境下运行的命令，将以>开头，例如 **>free**。

第1章 内存的测量

关于系统内存使用，笔者将按照明确目标→寻找评估方法，了解当前状况→对系统内存使用进行优化→重新测量，评估改善状况的过程，来阐述系统的内存使用与优化。

第一步，明确目标，针对系统内存优化，目标有两个：

（1）每个守护进程使用的内存要尽可能少，这样系统的剩余内存才会多，才能够支撑更多的应用同时在系统内运行。

（2）系统在长时间运行之后，各个守护进程仍然保持着较低的内存使用。这就要求进程在运行过程中没有内存泄漏，在长时间运行之后，系统仍能保持较多的剩余内存。

本章将重点放在第二步：寻找评估方法，了解当前状况。接下来的第 2、3、4 章将重点放在第三步对系统内存使用优化上，其中第 2 章进程内存优化和第 3 章系统内存优化将主要针对目标 1，讲述如何优化内存使用。第 4 章内存泄漏将主要针对目标 2，讲述如何发现和修正系统内存泄漏。

现在回到第二步，当前最重要的任务是了解系统有多少内存可以使用。如果是在 Windows 平台，可以很方便地从任务管理器中获得当前系统有多少可用内存、每个进程的内存占用量，但在 Linux 中该从何处获得这些信息呢？

1.1 系统当前可用内存

可以在 Linux 中敲入 free 命令获得当前系统的内存使用情况。

注意：笔者所使用的嵌入式 Linux 系统中集成了 Busybox，故在本书中所使用的很多命令的前面都加了“busybox”。

```
#busybox free
             total         used         free       shared      buffers
Mem:         55636        52808         2828            0         3132
Swap:            0            0            0
Total:       55636        52808         2828
```

当笔者满心欢喜地敲入上面的命令时，收到的却是一阵阵的冷汗，2828KB，系统只剩下了 2MB 多的内存，系统还能跑吗，是不是命令有错误？

来看看在 PC 上的 Linux 系统中，运行结果如何。

```
>free
             total         used         free       shared      buffers       cached
Mem:       4091524      4021016        70508            0         7656      1824312
-/+ buffers/cache:      2189048      1902476
```

Swap: 4088532 2891732 1196800

这里先解释几个关键的概念。

buffers：主要是用来给 Linux 系统中块设备做缓冲区。

cached：用来缓冲打开的文件。

在系统中内存是很宝贵的资源，故 Linux 在内存使用上的宗旨是：如果内存充足，不用白不用，尽量使用内存来缓存一些文件，从而加快进程的运行速度；而当内存不足时，这些内存又会被回收，供程序使用。

所以真正可用的内存=free+buffers+cached=70508+7656+1824312=1902476KB。

这样就很清楚了，的确是 busybox 在实现 free 时有缺陷，它缺少了关键的一列 cached，因此无法获得确切的空闲内存值。

既然 free 命令行不通，那么重点放在了 free 的数据是从哪里来的。

答案是：/proc。/proc 目录是一个特殊的文件系统，它不占用磁盘空间，该目录下的内容是根据用户请求的信息，由 Linux 内核实时生成的，所以可以通过 proc 目录，来获取 Linux 内核中的一部分数据。

可以从 proc 目录下的 meminfo 文件了解当前系统的内存使用情况。

 # cat /proc/meminfo

MemTotal:	55880 KB
MemFree:	2252 KB
Buffers:	3760 KB
Cached:	26112 KB
SwapCached:	0 KB
Active:	34652 KB
Inactive:	8716 KB
HighTotal:	0 KB
HighFree:	0 KB
LowTotal:	55880 KB
LowFree:	2252 KB
SwapTotal:	0 KB
SwapFree:	0 KB
Dirty:	0 KB

从上面的结果，可以知道系统当前总共拥有 55880KB 物理内存，注意这 55880KB 的物理内存包含了 Linux 内核自身所包含的物理内存，所以实际用户态程序能够使用的内存要少于 55880KB。其中 MemFree = 2252KB，Buffers = 3760 KB，Cached = 26112 KB。

那么当前可用的物理内存=Memfree+Buffers+Cached=2252+3760+26112=32124KB，应该说还是不少的。现在就可以很容易地计算出当前系统所使用的物理内存：

使用的物理内存=系统内存-可用内存=55880-32124=23756KB

下面的问题便是，这 23MB 的内存都是被谁使用了呢？

1.2 进程的内存使用

有了上面的经验，这次很自然地又想到了 proc 目录。进到 proc 目录下，查看一下都有什么内容。

```
# cd /proc
# ls
```

1	298	7
10	299	72
100	3	8
……	……	……
14	345	emg
159	381	kallsyms
16	384	kmsg
161	388	loadavg

这些数字，代表着一个个目录，同时这些数字也与当前系统中运行进程的 PID 一一对应，在这些目录下面的文件，记录着这些进程在 Linux 内核中相应的数据。

在介绍相关文件之前，笔者不得不先停下来，介绍一个概念：“虚拟内存”。

1.2.1 虚拟内存与物理内存

程序员编写程序时，不必去考虑物理内存，在 32 位的操作系统中，面对的是每个进程 4GB 的内存空间。而实际上呢，一台拥有 64MB 物理内存的设备上，可能要跑着几十个、甚至上百个这样的进程。

我们将程序员所面对的 4GB 内存空间，称为虚拟内存，操作系统为程序员屏蔽了物理内存的使用。在 Linux 中采用了延迟分配物理内存的策略，针对进程的内存分配请求，它只是在内核中分配一段虚拟地址，只有当确实使用这块内存时，系统才会为其分配物理地址。

下面来看一段代码：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main() {
5          char *p =(char *) malloc(10);
6          char *p1=（char *） malloc(200);
7          strcpy(p,"123");
8          return 0;
9  }
```

问题一：虚拟内存、物理内存与代码的对应关系是什么？

在 char *p=(char *)malloc(10)，只是分配了虚拟内存，内核不会分配物理页面给进程。

在 strcpy(p,"123")；进程需要使用这块内存了，内核会产生一个页故障，从而为系统分配一个物理页面。

因此，在执行完第 7 行之后，系统为指针 p1 只是分配了一个虚拟空间，而为指针 p 分配了相应的物理内存。

问题二：虚拟内存与物理内存有多大？

虚拟内存是 210B，物理内存是 4KB，因为内核分配物理内存的最小单位为一个物理页面，一个物理页面为 4KB。

1.2.2 /proc/{pid}

在介绍了物理内存和虚拟内存的区别之后，笔者继续介绍在 proc 目录下有关进程的文件。为了能够让大家更清楚地了解/proc/{pid}目录下的文件，在这里写一个简单的例子，编译后在系统中运行，然后查看 proc 目录下对应的信息。

```
hello.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int main() {
    char *p=(char *)malloc(20);
    strcpy(p,"123");
    pid_t pid=getpid();
    printf("pid:%d\n",pid);
    pause();                    //主要是为了进程停在这里，以便查看相关信息。
    return 0;
}
```

使用 GCC 编译后，下载到设备上。

>gcc –o hello hello.c

在设备上运行：

#./hello

pid:491

现在进程停在这里，可以另外起一个 telnet 窗口，到 proc 目录下去查看其信息。

#cd /proc/491

 # ls

attr auxv cmdline cwd environ exe fd maps
memmap mounts nodemap root stat statm status task wchan

注意：在 Linux 的不同版本中，其下的文件会有所不同。

在这里只介绍几个与内存相关的文件(statm、maps、memmap)，其余文件的信息大家可以在互联网上查到。

#cat statm

345 87 74 1 0 58 0

这里有 7 个数，它们以页（4KB）为单位，下面分别介绍它们的含义。

345：　Size，任务虚拟地址空间的大小。

87：　Resident，应用程序正在使用的物理内存的大小。

74：　Shared，共享页数。

1：　Trs，程序所拥有的可执行虚拟内存的大小。

0：　Lrs，被映像到任务的虚拟内存空间的库的大小。

58：　Drs，程序数据段和用户态的栈的大小。

0：　dt，脏页数量（已经修改的物理页面）。

其中 Size、Trs、Lrs、Drs 对应于进程的虚拟内存，Resident、shared、dr 对应于物理内存。

注意：在笔者所使用的 Linux 内核中，dt 的代码被修改了，直接返回 0。

从 statm 的数值可以得知，如此简单的一个程序，运行时虚存居然占了 345 个页面，也就是 345*4=1380KB 内存，大大超出了想象。

下面就来看看进程为什么使用了 1380KB 的内存：

#cat maps

00008000-00009000	r-xp	00000000	1f:12	288	/mnt/msc_int0/hello
00010000-00011000	rw-p	00000000	1f:12	288	/mnt/msc_int0/hello
00011000-00032000	rwxp	00011000	00:00	0	
40000000-40002000	rw-p	40000000	00:00	0	
41000000-41017000	r-xp	00000000	1f:0d	817360	/lib/ld-2.3.3.so
4101e000-41020000	rw-p	00016000	1f:0d	817360	/lib/ld-2.3.3.so
41028000-41120000	r-xp	00000000	1f:0d	817593	/lib/libc-2.3.3.so
41120000-41128000	---p	000f8000	1f:0d	817593	/lib/libc-2.3.3.so
41128000-41129000	r--p	000f8000	1f:0d	817593	/lib/libc-2.3.3.so
41129000-4112c000	rw-p	000f9000	1f:0d	817593	/lib/libc-2.3.3.so
4112c000-4112e000	rw-p	4112c000	00:00	0	
befeb000-bf000000	rwxp	befeb000	00:00	0	

下面以第一行为例，解释各列的内容。

第一列：00008000-00009000，代表该内存段的虚拟地址。

第二列：r-xp，代表着该段内存的权限，其值含义为：r=读，w=写，x=执行，s=共享，p=私有。

第三列：00000000，代表在进程里地址的偏移量。

第四列：1f:12，映射文件的主设备号和次设备号。可以通过“cat /proc/devices”来查看设备信息：

 # cd /proc

 # cat devices

Character devices:

1 mem

4 /dev/vc/0

5 /dev/tty

```
.....
252 mxc_ipc
254 devfs
Block devices:
  1 ramdisk
  7 loop
 31 mtdblock
243 mmc
```

1f转换成10进制为31，因此该段内存映射位于mtdblock设备的文件。

第五列：288，映像文件的节点号。

第六列：/mnt/msc_int0/hello，映像文件的路径，正好对应着执行文件所对应的目录。从maps中可以知道，在进程的内存空间中，不光包括执行文件本身，还包括ld-2.3.3.so和libc-2.3.3.so两个动态库。

在这里，先抛开动态库所使用的内存不提，只关注执行文件自身内存的使用。

问题一：下面各行都是什么含义？

```
00008000-00009000    r-xp    00000000    1f:12    288    /mnt/msc_int0/hello
00010000-00011000    rw-p    00000000    1f:12    288    /mnt/msc_int0/hello
00011000-00032000    rwxp    00011000    00:00    0
40000000-40002000    rw-p    40000000    00:00    0
4112c000-4112e000    rw-p    4112c000    00:00    0
befeb000-bf000000    rwxp    befeb000    00:00    0
```

回答：

```
00008000-00009000    r-xp    00000000    1f:12    288    /mnt/msc_int0/hello
```

从r-xp可以得知其权限为只读、可执行，该段内存地址对应于执行文件的代码段，程序的代码也需要加载到内存才可以执行的。

```
00010000-00011000    rw-p    00000000    1f:12    288    /mnt/msc_int0/hello
```

从rw-p可以得知其权限为可读写、不可执行，该段内存地址对应于执行文件的数据段，主要存储执行文件所用到的全局变量、静态变量。

```
00011000-00032000    rwxp    00011000    00:00    0
```

从rwxp可以得知其权限是可读写、可执行，内存地址向上增长，而且不对应文件，其为堆段，进程使用malloc申请的内存都放在这里。

00011000-00032000这段内存地址空间有多大呢？132KB。为什么代码只申请了20个字节，而系统却为其分配了132KB虚拟内存呢？

这个问题，在后面专门讲堆段时会提到。

```
40000000-40002000    rw-p    40000000    00:00    0
```

从权限rw-p来看，像数据段，但没有映射的文件；从没有映射文件这一条来看，像堆段，但其权限又不对，那么它是什么呢？

这个问题，现在笔者也回答不上来，往后看吧！

```
befeb000-bf000000    rwxp    befeb000    00:00    0
```

其位于地址的顶端，内存区域向下增长，这段内存为栈段。这里可能有人要问，为什么

堆和栈的内存权限是可执行的呢？

关于堆栈段是否具有代码执行的能力，Linux 的版本并没有严格的规定，有些是可以在堆栈中加一些可执行的代码的。

问题二：在 32 位操作系统中，每个进程拥有 4GB 的虚拟内存空间，既然栈是从顶端向下增长，那么栈顶的地址应该是 0xffffffff，而不是 0xbf000000。

先看看 0xbf000000 是多少？转化成十进制为 3 204 448 256，不多不少整整 3GB。

每个进程通过系统调用访问内核，Linux 内核空间由系统内的所有进程共享。从进程的角度来看，每个进程拥有 4GB 的虚拟地址空间。其中 0～3GB 为各个进程的私有用户空间，这个空间对系统中的其他进程是不可见的，最高的 1GB 内核空间则为所有进程以及内核所共享，如图 1-1 所示。

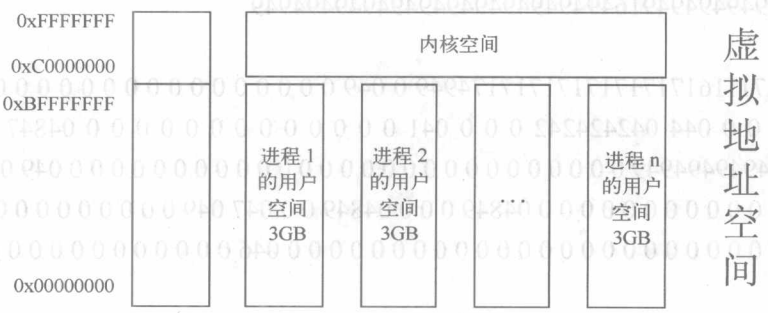

图 1-1　Linux 中进程的虚拟内存空间

虚拟内存一共 4GB，分为两部分：

- 内核空间（最高的 1GB）。
- 用户空间（较低的 3GB）。

因此，更确切地说，每个进程最大拥有 3GB 私有虚拟内存空间。

问题三：堆段地址是向上增长，进程可以通过扩展堆顶地址来增加堆所占用的内存，而在 maps 中：

```
00011000-00032000    rwxp    00011000 00:00 0    堆地址
40000000-40002000    rw-p    40000000 00:00 0    共享库的一段地址
```

如果堆地址一直增长到 40000000 的话，堆段该如何处理呢？

这的确是个问题，这里笔者先埋下一个伏笔，在后面讲到进程堆段内存管理时，会详细说明。

现在笔者来回答进程 hello 为什么会占用了 1380KB 内存的问题。关键在于：

```
41000000-41017000 r-xp 00000000 1f:0d 817360    /lib/ld-2.3.3.so    (92KB)
41028000-41120000 r-xp 00000000 1f:0d 817593    /lib/libc-2.3.3.so  (992KB)
```

也就是说，仅这两个库的代码段就占用了 92+992=1084KB 内存，因此从 statm 得出来的内存占用量 1380KB，包含了其依赖动态库占用的内存，而可执行文件本身并没有使用这么多内存。

以上涉及的全部为虚拟内存的概念，不涉及物理内存的使用。下面就着手了解一个进程的物理内存使用情况。