SUSE Linux Enterprise Desktop 15 SP7

系统分析和微调指南

系统分析和微调指南

SUSE Linux Enterprise Desktop 15 SP7

本指南可帮助管理员检测和解决问题并优化系统。

出版日期：2026 年 4 月 30 日

https://documentation.suse.com

目录

前言 xiii

1 可用文档 xiii

2 改进文档 xiii

3 文档约定 xiv

4 支持 xvi

SUSE Linux Enterprise Desktop 支持声明 xvi • 技术预览 xvii

I 基础 1

1 有关系统微调的一般说明 2

1.1 确认要解决的问题 2

1.2 排除常见问题 3

1.3 确定瓶颈 4

1.4 逐步微调 4

II 系统监控 5

2 系统监控实用程序 6

2.1 多用途工具 7

vmstat 7 • **dstat** 10 • 系统活动信息：**sar** 11

2.2 系统信息 16

设备负载信息：**iostat** 16 • 处理器活动监控：**mpstat** 17 • 处理器频率监控：**turbostat** 18 • 任务监控：**pidstat** 18 • 内核环形缓冲区：**dmesg** 19 • 打开的文件的列表：**lsof** 19 • 内核和 udev 事件序列查看器：**udevadm monitor** 20

2.3 流程 21
进程间通讯：**ipcs** 21 · 进程列表：**ps** 21 · 进程树：**pstree** 23 · 进程表：**top** 24 · IBM Z 超级管理程序监控器：**hyptop** 25 · 顶层式 I/O 监控器：**iotop** 27 · 修改进程的资源优先级：**nice** 和 **renice** 28

2.4 内存 29
内存用量：**free** 29 · 内存使用详情：/proc/meminfo 29 · 进程内存用量：smaps 34 · numaTOP 34

2.5 网络 35
基本网络诊断：**ip** 35 · 显示进程的网络用量：**nethogs** 36 · 以太网卡细节：ethtool 36 · 显示网络状态：**ss** 37

2.6 /proc 文件系统 39
procinfo 42 · 系统控制参数：/proc/sys/ 43

2.7 硬件信息 44
PCI 资源：**lspci** 44 · USB 设备：**lsusb** 45 · 监控和微调热子系统：**tmon** 46 · MCELog：计算机检查异常 (MCE) 47 · AMD64/Intel 64：dmidecode：DMI 表解码器 48 · POWER：列出硬件 49

2.8 文件和文件系统 49
确定文件类型：**file** 49 · 文件系统及其用法：**mount**、**df** 和 **du** 49 · 有关 ELF 二进制文件的其他信息 50 · 文件属性：**stat** 51

2.9 用户信息 52
访问文件的用户：**fuser** 52 · 哪些用户在执行哪些操作：**w** 52

2.10 时间和日期 53
使用 **time** 计量时间 53

2.11 绘制数据图表：RRDtool 54
RRDtool 的工作原理 54 · 实际应用示例 55 · 更多信息 59

3 系统日志文件 60

3.1 /var/log/ 中的系统日志文件 60

3.2 查看和分析日志文件 62

3.3 使用 **logrotate** 管理日志文件 63

3.4 使用 **logwatch** 监控日志文件 64

3.5 为 root 配置邮件转发 65

3.6 将日志消息转发到中心系统日志服务器 66

设置中心系统日志服务器 67 • 设置客户端计算机 68 • 更多信息 69

3.7 使用 **logger** 创建系统日志项 69

III 内核监控 70

4 SystemTap — 过滤和分析系统数据 71

4.1 概念概述 71

SystemTap 脚本 71 • Tapset 72 • 命令和特权 72 • 重要文件和目录 73

4.2 安装和设置 73

4.3 脚本语法 75

探测格式 76 • SystemTap 事件（探测点） 77 • SystemTap 处理程序（探测主体） 78

4.4 示例脚本 82

4.5 用户空间探测 83

4.6 更多信息 84

5 内核探测 85

5.1 支持的体系结构 85

5.2 内核探测的类型 86
Kprobe 86 • Jprobe 86 • 返回探测 86
5.3 Kprobe API 87
5.4 debugfs 界面 87
列出已注册的内核探测 88 • 全局启用/禁用内核探测 88
5.5 更多信息 88
6 使用 Perf 进行基于硬件的性能监控 90
6.1 基于硬件的监控 90
6.2 采样和计数 90
6.3 安装 Perf 91
6.4 Perf 子命令 91
6.5 统计特定类型的事件 92
6.6 记录特定于特定命令的事件 92
6.7 更多信息 93
7 OProfile — 系统范围的分析器 95
7.1 概念概述 95
7.2 安装和要求 95
7.3 可用的 OProfile 实用程序 95
7.4 使用 OProfile 96
创建报告 96 • 获取事件配置 97
7.5 生成报告 99
7.6 更多信息 99

8 动态调试 - 内核调试消息 101

8.1 动态调试的优势 101

8.2 检查动态调试的状态 101

8.3 使用动态调试 102

8.4 查看动态调试消息 103

IV 资源管理 104

9 一般系统资源管理 105

9.1 规划安装 105

分区 105 • 安装范围 105 • 默认目标 106

9.2 禁用不必要的服务 106

9.3 文件系统和磁盘访问 107

文件系统 107 • 时戳更新策略 108 • 使用 **ionice** 指定磁盘访问优先级 108

10 内核控制组 110

10.1 概述 110

混合 cgroup 层次结构 110

10.2 资源统计 111

10.3 设置资源限制 111

10.4 使用 TasksMax 防止派生炸弹 111

查找当前的默认 TasksMax 值 112 • 覆盖 DefaultTasksMax 值 112 • 针对用户的默认 TasksMax 限制 114

10.5 使用 cgroup 进行 I/O 控制 114

先决条件 114 • 配置控制数量 116 • I/O 控制行为和设置预期 116 • 用户会话中的资源控制 117

10.6 更多信息 118

11 自动平衡非一致性内存访问 (NUMA) 119

11.1 实现 119

11.2 配置 119

11.3 监控 121

11.4 影响 121

12 电源管理 124

12.1 CPU 级别的电源管理 124

C 状态（处理器运行状态） 124 • P 状态（处理器性能状态） 125 • Turbo 功能 126

12.2 内核中调节器 126

12.3 cpupower 工具 127

使用 **cpupower** 查看当前设置 127 • 使用 **cpupower** 查看内核空闲统计数据 128 • 使用 **cpupower** 监控内核和硬件统计数据 129 • 使用 **cpupower** 修改当前设置 130

12.4 特殊微调选项 131

P 状态的微调选项 131

12.5 查错 131

12.6 更多信息 132

12.7 使用 powerTOP 监控能耗 132

V 内核微调 135

13 微调 I/O 性能 136

13.1 切换 I/O 调度 136

13.2 使用 blk-mq I/O 路径的可用 I/O 电梯算法 136
MQ-DEADLINE 137 • NONE 138 • BFQ（预算公平队列） 138 • KYBER 139

13.3 I/O 屏障微调 140

14 微调任务调度程序 141

14.1 简介 141
抢占 141 • 时间片 141 • 进程优先级 142

14.2 进程分类 142

14.3 完全公平调度程序 143
CFS 的工作原理 143 • 将进程分组 144 • 内核配置选项 144 • 术语 144 • 使用 **chrt** 更改进程的实时属性 145 • 使用 **sysctl** 进行运行时微调 146 • 调试界面和调度程序统计数据 150

14.4 更多信息 152

15 微调内存管理子系统 153

15.1 内存用量 153
匿名内存 153 • 页缓存 154 • 缓冲区缓存 154 • 缓冲区头 154 • 写回 154 • 预读 155 • VFS 缓存 155

15.2 减少内存用量 155
减少 malloc（匿名）用量 155 • 减少内核内存开销 156 • 内存控制器（内存 cgroup） 156

15.3 虚拟内存管理器 (VM) 可调参数 156
回收比率 156 • 写回参数 157 • SUSE Linux Enterprise 12 与 SUSE Linux Enterprise 11 之间的 I/O 写入时间差异 159 • 预读参数 160 • 透明大页参数 160 • khugepaged 参数 161 • 其他 VM 参数 162

15.4 监控 VM 行为 162

16 微调网络 164

16.1 可配置的内核套接字缓冲区 164

16.2 检测网络瓶颈和分析网络流量 166

16.3 Netfilter 166

16.4 使用接收包导向 (RPS) 改善网络性能 166

VI 处理系统转储 169

17 跟踪工具 170

17.1 使用 strace 跟踪系统调用 170

17.2 使用 ltrace 跟踪库调用 174

17.3 使用 Valgrind 进行调试和分析 176

安装 176 • 支持的体系结构 176 • 一般信息 176 • 默认选项 177 • Valgrind 的工作原理 178 • 消息 178 • 错误消息 180

17.4 更多信息 181

18 Kexec 和 Kdump 182

18.1 简介 182

18.2 所需的软件包 182

18.3 Kexec 内部结构 183

18.4 计算 crashkernel 分配大小 184

18.5 Kexec 基本用法 187

18.6 如何为例程重引导配置 Kexec 188

18.7 Kdump 基本配置 188

Kdump 的手动配置 189 • YaST 配置 190 • Kdump 通过 SSH 传输 192

18.8 分析崩溃转储 193
内核二进制文件格式 194
18.9 Kdump 高级配置 197
18.10 更多信息 198
19 使用 systemd-coredump 针对应用程序崩溃进行调试 200
19.1 用法和配置 200
VII 使用精确时间协议同步时钟 204
20 精确时间协议 205
20.1 PTP 简介 205
PTP Linux 实现 205
20.2 使用 PTP 206
网络驱动程序和硬件支持 206 • 使用 ptp4l 207 • ptp4l 配置文件 208 • 延迟测量 208 • PTP 管理客户端：pmc 209
20.3 使用 phc2sys 同步时钟 210
校验时间同步 211
20.4 配置示例 212
20.5 PTP 和 NTP 213
NTP 到 PTP 的同步 213 • 配置 PTP-NTP 网桥 214
A GNU licenses 215

前言

1 可用文档

联机文档

可在 https://documentation.suse.com 上查看我们的联机文档。您可浏览或下载各种格式的文档。

注意：最新更新

最新的更新通常会在本文档的英文版中提供。

SUSE 知识库

如果您遇到问题，请参考 https://www.suse.com/support/kb/ 上提供的联机技术信息文档 (TID)。在 SUSE 知识库中搜索根据客户需求提供的已知解决方案。

发行说明

有关发行说明，请参见 https://www.suse.com/releasenotes/。

在您的系统上

如需脱机使用，您也可在系统的 `/usr/share/doc/release-notes` 下找到该发行说明。各软件包的相应文档可在 `/usr/share/doc/packages` 中找到。

许多命令的**手册页**中也对相应命令进行了说明。要查看手册页，请运行 `man` 后跟特定的命令名。如果系统上未安装 `man` 命令，请使用 `sudo zypper install man` 加以安装。

2 改进文档

欢迎您提供针对本文档的反馈及改进建议。您可以通过以下渠道提供反馈：

服务请求和支持

有关产品可用的服务和支持选项，请参见 https://www.suse.com/support/。

要创建服务请求，需在 SUSE Customer Center 中注册订阅的 SUSE 产品。请前往 https://scc.suse.com/support/requests 并登录，然后单击新建。

Bug 报告

在 https://bugzilla.suse.com/ 中报告文档问题。

要简化此过程，请点击本文档 HTML 版本中标题旁边的报告问题图标。这样会在 Bugzilla 中预先选择正确的产品和类别，并添加当前章节的链接。然后，您便可以立即开始键入 Bug 报告。

需要一个 Bugzilla 帐户。

贡献

要帮助改进本文档，请点击本文档 HTML 版本中标题旁边的 Edit Source document（编辑源文档）图标。然后您会转到 GitHub 上的源代码，可以在其中提出拉取请求。

需要一个 GitHub 帐户。

注意：Edit source document（编辑源文档）仅适用于英语版本

Edit source document（编辑源文档）图标仅适用于每个文档的英语版本。对于所有其他语言，请改用报告问题图标。

有关用于本文档的文档环境的详细信息，请参见储存库的 README。

邮件

您也可以将有关本文档的错误以及反馈发送至 `doc-team@suse.com`。请在其中包含文档标题、产品版本和文档发布日期。此外，请包含相关的章节号和标题（或者提供 URL），并提供问题的简要说明。

3 文档约定

本文档中使用了以下通知和排版约定：

- `/etc/passwd`：目录名称和文件名
- `PLACEHOLDER`：请将 `PLACEHOLDER` 替换为实际值
- `PATH`：环境变量
- **`ls`**、`--help`：命令、选项和参数

- `user`：用户或组的名称
- `package_name`：软件包的名称
- `Alt`、`Alt`–`F1`：按键或组合键。按键以大写字母显示，与键盘上的一样。
- 文件、文件 › 另存为：菜单项、按钮
- Chapter 1, “Example chapter”：对本指南中其他章节的交叉引用。
- 必须使用 `root` 特权运行的命令。您还可以在这些命令前加上 **`sudo`** 命令，以非特权用户身份来运行它们：

```
# command
> sudo command
```

- 非特权用户也可以运行的命令：

```
> command
```

- 可以通过一行末尾处的反斜线字符 (\) 拆分成两行或多行的命令。反斜线告知外壳命令调用将会在该行末尾后面继续：

```
> echo a b \
c d
```

- 显示命令（前面有一个提示符）和外壳返回的相应输出的代码块：

```
> command
output
```

- 注意事项

警告：警报通知

在继续操作之前，您必须了解的关键性信息。向您指出有关安全问题、潜在数据丢失、硬件损害或物理危害的警告。

重要：重要通知

在继续操作之前，您必须了解的重要信息。

注意：注意通知

额外信息，例如有关软件版本差异的信息。

提示：提示通知

有用信息，例如指导方针或实用性建议。

- 精简通知

额外信息，例如有关软件版本差异的信息。

有用信息，例如指导方针或实用性建议。

4 支持

下面提供了 SUSE Linux Enterprise Desktop 的支持声明和有关技术预览的一般信息。有关产品生命周期的细节，请参见 https://www.suse.com/lifecycle。

如果您有权获享支持，可在 https://documentation.suse.com/sles-15/html/SLES-all/cha-adm-support.html 中查找有关如何收集支持票据所需信息的细节。

4.1 SUSE Linux Enterprise Desktop 支持声明

要获得支持，您需要订阅适当的 SUSE 产品。要查看为您提供的具体支持服务，请前往 https://www.suse.com/support/ 并选择您的产品。

支持级别的定义如下：

L1

问题判定，该技术支持级别旨在提供兼容性信息、使用支持、持续维护、信息收集，以及使用可用文档进行基本查错。

L2

问题隔离，该技术支持级别旨在分析数据、重现客户问题、隔离问题区域，并针对级别 1 不能解决的问题提供解决方法，或完成准备工作以提交级别 3 处理。

L3

问题解决，该技术支持级别旨在借助工程方法解决级别 2 支持所确定的产品缺陷。

对于签约客户与合作伙伴，SUSE Linux Enterprise Desktop 包含除以下项目外的其他所有软件包的 L3 支持：

- 技术预览。
- 声音、图形、字体和作品。
- 需要额外客户合同的软件包。
- 名称以 `-devel` 结尾的软件包（包含头文件和类似的开发人员资源）只能与其主软件包一起获得支持。

SUSE 仅支持使用原始软件包，即，未发生更改且未重新编译的软件包。

4.2 技术预览

技术预览是 SUSE 提供的旨在让用户大致体验未来创新的各种软件包、堆栈或功能。随附这些技术预览只是为了提供方便，让您有机会在自己的环境中测试新的技术。非常希望您能提供反馈。如果您测试了技术预览，请联系 SUSE 代表，将您的体验和用例告知他们。您的反馈对于我们的未来开发非常有帮助。

技术预览存在以下限制：

- 技术预览仍处于开发阶段。因此，它们可能在功能上不完整、不稳定，或者**不**适合生产用途。
- 技术预览**不**受支持。

- 技术预览可能仅适用于特定的硬件体系结构。
- 技术预览的细节和功能可能随时会发生变化。因此，可能无法升级到技术预览的后续版本，而只能进行全新安装。
- SUSE 可能会发现某个预览不符合客户或市场需求，或者未遵循企业标准。技术预览可能会随时从产品中删除。SUSE 不承诺未来将提供此类技术的受支持版本。

如需大致了解产品随附的技术预览，请参见 https://www.suse.com/releasenotes 上的发行说明。

I 基础

1 有关系统微调的一般说明 2

1 有关系统微调的一般说明

本手册介绍如何查明导致性能问题的原因，并提供这些问题的解决方法。在您开始微调系统之前，应确保已排除常见问题并已找到问题的原因。另外，还应针对如何微调系统制定详细的计划，因为遵循随机出现的微调提示往往无助于解决问题，甚至可能让情况变得更糟。

过程 1.1： 微调系统时的一般方法

1. 指定需要解决的问题。
2. 如果性能下降是近期才出现的问题，请确定最近对系统所做的任何更改。
3. 确定将问题视为性能问题的原因。
4. 指定可用于分析性能的度量。例如，此指标可以是延迟、吞吐量、同时登录用户最大数量或活动用户最大数量。
5. 使用上一步骤中指定的指标来测量当前性能。
6. 确定应用程序将大部分时间消耗在哪些子系统上。
7.
   a. 监控系统和/或应用程序。
   b. 分析数据，对消耗时间的子系统进行分类。
8. 微调在上一步骤中确定的子系统。
9. 使用前面相同的度量，在不进行监控的情况下重新测量当前性能。
10. 如果性能仍不可接受，请从步骤 3 重新开始。

1.1 确认要解决的问题

在开始微调系统之前，请尝试尽可能具体地描述问题。“系统速度慢！”这样的问题说明没有帮助。例如，要对是需要在一般情况下提高系统速度还是仅在高峰期提高进行区分。

此外，应确保您能够对问题应用测量，否则无法校验微调是否成功。您应始终能对“微调前”和“微调后”的状态进行对比。要使用的指标取决于您正在调查的方案或应用程序。例如，相关的 Web 服务器指标可以通过以下方面表示：

延迟

传送页所需的时间

吞吐量

每秒提供的页数量，或每秒传送的 MB 数量

活动用户

能以可接受的延迟在下载页的同时仍接收页的最大用户数

1.2 排除常见问题

性能问题往往由网络或硬件问题、bug 或配置问题造成。在尝试微调系统之前，请确保排除下面所列的此类问题：

- 检查 `systemd` 日记的输出（请参见《管理指南》, 第 21 章 “**`journalctl`**：查询 `systemd` 日记”）中的异常项。
- 检查（使用 **`top`** 或 **`ps`**）是否有特定进程行为异常，不正常消耗 CPU 时间或内存。
- 通过检查 `/proc/net/dev` 来检查网络问题。
- 如果物理磁盘出现 I/O 问题，请确保其并非由硬件问题（使用 `smartmontools`）或整个磁盘导致。
- 确保将后台作业安排在服务器负载较低的时段执行。此外，应以较低的优先级（通过 **`nice`** 设置）运行这些作业。
- 如果计算机运行有使用相同资源的多个服务，请考虑将服务移到另一台服务器。
- 最后，确保您的软件是最新的。

1.3 确定瓶颈

微调系统时，最困难的是如何确定瓶颈。SUSE Linux Enterprise Desktop 提供了许多工具来帮助您完成此任务。有关一般系统监控应用程序和日志文件分析的详细信息，请参见第 II 部分 “系统监控”。如果问题需要长时间的深入分析，Linux 内核会提供执行此类分析的方法。有关具体范围，请参见第 III 部分 “内核监控”。

收集数据后，需要对其进行分析。首先，检查服务器的硬件（内存、CPU、总线）及其 I/O 容量（磁盘、网络）是否足够。如果满足这些基本条件，则微调系统可能是有益的。

1.4 逐步微调

确保认真规划微调本身。请务必每次仅执行一个步骤。只有这样，才能测量所做的更改是起到了改善作用还是进一步产生了负面影响。应对每个微调活动进行足够长时间的测量，确保可以基于重要数据执行分析。如果您未测量到正面影响，请不要做出永久性更改。否则可能会在将来造成负面影响。

II 系统监控

2 系统监控实用程序 6

3 系统日志文件 60

2 系统监控实用程序

您可以使用各种程序、工具和实用程序来检查系统的状态。本章介绍其中的一部分以及它们最重要且最常用的参数。

注意： 使用 **supportconfig** 收集和分析系统信息

除了下面介绍的实用程序外，SUSE Linux Enterprise Desktop 还包含 **supportconfig**，此工具可创建有关系统的报告，例如当前内核版本、硬件、已安装软件包、分区设置等等。这些报告用于向 SUSE 支持人员提供创建支持票据时所需的信息。但是，还可根据这些报告分析已知问题，以帮助快速解决问题。为此，SUSE Linux Enterprise Desktop 提供了一个设备和一个命令行工具用于进行 Supportconfig 分析 (SCA)。有关详细信息，请参见《管理指南》, 第 41 章 “收集系统信息以供支持所用”。

对于所述的每条命令，将提供相关输出的示例。在示例中，第一行是命令本身（在 `tux >` 或 `root #` 之后）。使用方括号 (`[...]`) 表示省略，必要时对较长的行进行换行。较长的行的换行符由反斜线 (`\`) 表示。

```
> command -x -y
output line 1
output line 2
output line 3 is annoyingly long, so long that \
    we need to break it
output line 4
[...]
output line 98
output line 99
```

我们尽量使说明简洁明了，以便包含尽可能多的实用程序。您可以在手册页中找到所有命令的更多信息。大多数命令还可识别参数 `--help`，该参数可生成可能参数的简要列表。

2.1 多用途工具

大部分 Linux 系统监控工具只会监控系统的某个方面，不过，也有几个工具的使用范围更广。要了解概况并确定想要进一步检查系统的哪个部分，请先使用这些工具。

2.1.1 vmstat

vmstat 可收集有关进程、内存、I/O、中断和 CPU 的信息：

```
vmstat [options] [delay [count]]
```

如果在不指定延迟和计数值的情况下调用此工具，它将显示自上次重引导以来的平均值。如果结合使用延迟值（以秒为单位）调用，它将显示给定时间段（以下示例中为两秒）的值。计数值指定 vmstat 应执行的更新次数。如果不指定此值，此工具会一直运行，直到用户手动将其停止。

例 2.1： 低负载计算机上的 vmstat 输出

```
> vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0  44264  81520    424 935736    0    0    12    25   27   34  1  0 98  0
 0
 0  0  44264  81552    424 935736    0    0     0     0   38   25  0  0 100  0
 0
 0  0  44264  81520    424 935732    0    0     0     0   23   15  0  0 100  0
 0
 0  0  44264  81520    424 935732    0    0     0     0   36   24  0  0 100  0
 0
 0  0  44264  81552    424 935732    0    0     0     0   51   38  0  0 100  0
 0
```

例 2.2： 高负载计算机（CPU 密集型）上的 vmstat 输出

```
> vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- -----
cpu------
 r  b   swpd   free   buff   cache   si   so    bi    bo   in   cs us sy id wa
 st
```

```
32  1  26236 459640 110240 6312648    0    0  9944     2 4552 6597 95  5  0  0
 0
23  1  26236 396728 110336 6136224    0    0  9588     0 4468 6273 94  6  0  0
 0
35  0  26236 554920 110508 6166508    0    0  7684 27992 4474 4700 95  5  0  0
 0
28  0  26236 518184 110516 6039996    0    0 10830     4 4446 4670 94  6  0  0
 0
21  5  26236 716468 110684 6074872    0    0  8734 20534 4512 4061 96  4  0  0
 0
```

提示：输出的第一行

vmstat 输出的第一行始终显示自上次重引导以来的平均值。

各列显示以下信息：

r

显示处于可运行状态的进程数。这些进程正在执行，或正在等待有可用的 CPU 槽。如果此列中的进程数目持续高于可用 CPU 数目，可能表示 CPU 算力不足。

b

显示正在等待除 CPU 以外资源的进程数。如果此列中的数字较大，可能表示出现了 I/O 问题（网络或磁盘）。

swpd

当前使用的交换空间量 (KB)。

free

未使用的内存量 (KB)。

inact

最近未使用的可回收的内存。仅当结合使用参数 `-a`（建议使用）调用 **`vmstat`** 时，此列才可见。

active

最近已使用的且正常情况下不会回收的内存。仅当结合使用参数 `-a`（建议使用）调用 **`vmstat`** 时，此列才可见。

buff

RAM 中包含文件系统元数据的文件缓冲区缓存 (KB)。结合使用参数 `-a` 调用 **`vmstat`** 时，此列不可见。

超速缓存

RAM 中包含实际文件内容的页缓存 (KB)。结合使用参数 `-a` 调用 **`vmstat`** 时，此列不可见。

si / so

每秒从交换空间移到 RAM (`si`) 或者从 RAM 移到交换空间 (`so`) 的数据量 (KB)。如果 `so` 值在很长一段时间内都较大，可能表示应用程序正在泄漏内存，并且泄漏的内存正在换出。如果 `si` 值在很长一段时间内都较大，可能表示在很长时间内处于非活动状态的应用程序现在恢复了活动状态。如果 `si` 和 `so` 值在很长一段时间内都较大，则证明存在交换震荡，并可能表示需要在系统中安装更多 RAM，因为没有足够的内存可以容纳工作集。

bi

每秒从块设备收到的块数（例如，磁盘读取）。交换也会影响此处显示的值。块大小根据文件系统而异，但可以使用 stat 实用程序来确定。如果需要吞吐量数据，则可使用 iostat。

bo

每秒向块设备发送的块数（例如，磁盘写入）。交换也会影响此处显示的值。

in

每秒中断数。如果此值较大，可能表示 I/O 级别（网络和/或磁盘）较高，但也有可能是出于其他原因触发了此状态，例如，其他活动触发了处理器间中断。请务必同时检查 `/proc/interrupts` 以识别中断的来源。

cs

每秒环境切换次数。这是内核将内存中一个程序的可执行代码替换为另一程序的可执行代码的次数。

us

执行应用程序代码的 CPU 用量百分比。

sy

执行内核代码的 CPU 用量百分比。

id

CPU 空闲时间百分比。如果此值在很长一段时间内都为零，表示您的 CPU 正在满负荷工作。这并不一定表示存在问题 — 建议参考 r 和 b 列中的值来确定您的计算机是否具备足够的 CPU 算力。

wa

如果“wa”时间不为零，表示由于正在等待 I/O 而损失了吞吐量。此问题有时无法避免，例如，首次读取某个文件而后台写回无法跟上读取进度等情况。此外，还可能表示存在硬件瓶颈（网络或硬盘）。最后，这可能表示需要微调虚拟内存管理器（请参见第 15 章 “微调内存管理子系统”）。

st

从虚拟机挪用的 CPU 时间百分比。

有关更多选项，请参见 **vmstat** `--help`。

2.1.2 dstat

dstat 可以取代 **vmstat**、**iostat**、**netstat** 或 **ifstat** 等工具。**dstat** 实时显示有关系统资源的信息。例如，您可结合来自 IDE 控制器的中断数比较磁盘用量，或结合磁盘吞吐量（以相同的间隔）比较网络带宽。

默认情况下，其输出将显示在易于阅读的表格中。或者，可以生成 CSV（适合用作电子表格导入格式）输出。

dstat 以 Python 编写，可以通过插件来增强其功能。

下面是一般语法：

```
dstat [-afv] [OPTIONS..] [DELAY [COUNT]]
```

所有选项和参数都是可选的。如果未指定任何参数，dstat 将显示有关 CPU（`-c`、`--cpu`）、磁盘（`-d`、`--disk`）、网络（`-n`、`--net`）、分页（`-g`、`--page`），以及系统的中断和环境切换（`-y`、`--sys`）的统计数据；它无休止地每秒刷新一次输出：

```
# dstat
You did not select any stats, using -cdngy by default.
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
```

```
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0   0|  15k   44k|   0     0 |   0    82B| 148   194
  0   0 100   0   0   0|   0     0 |5430B  170B|   0     0 | 163   187
  0   0 100   0   0   0|   0     0 |6363B  842B|   0     0 | 196   185
```

`-a`、`--all`
: 等于 `-cdngy`（默认值）

`-f`、`--full`
: 扩展 `-C`、`-D`、`-I`、`-N` 和 `-S` 发现列表

`-v`、`--vmstat`
: 等于 `-pmgdsc`、`-D total`

DELAY
: 每次更新的延迟秒数

COUNT
: 在退出之前要显示的更新次数

默认延迟为 1，计数为未指定（无限制）。

有关详细信息，请参见 **dstat** 的手册页及其网页（网址为 http://dag.wieers.com/home-made/dstat/）。

2.1.3 系统活动信息：sar

sar 可以生成关于几乎所有重要系统活动的各种报告，其中包括 CPU、内存、IRQ 用量、I/O 和网络。它还可以实时生成报告。**sar** 命令从 `/proc` 文件系统收集数据。

注意：sysstat 软件包

sar 命令是 `sysstat` 软件包的一部分。请使用 YaST 或 **zypper in sysstat** 命令安装该软件包。`sysstat.service` 在默认情况下不会启动，必须使用以下命令来启用并启动它：

```
> sudo systemctl enable --now sysstat
```

2.1.3.1 使用 sar 生成报告

要实时生成报告，请调用 **sar** 并指定间隔（秒）和计数。要从文件生成报告，请使用选项 `-f` 指定文件名，而不要指定间隔和计数。如果未指定文件名、间隔和计数，**sar** 将尝试从 `/var/log/sa/saDD` 生成报告，其中 `DD` 表示当天。这是 **sadc**（系统活动数据收集器）将其数据写入到的默认位置。您可使用多个 `-f` 选项查询多个文件。

```
sar 2 10                          # real time report, 10 times every 2 seconds
sar -f ~/reports/sar_2014_07_17   # queries file sar_2014_07_17
sar                               # queries file from today in /var/log/sa/
cd /var/log/sa && \
sar -f sa01 -f sa02               # queries files /var/log/sa/0[12]
```

下面提供了有用的 **sar** 调用示例及其解释。有关每列含义的详细信息，请参见 **sar** 的 `man (1)`。

注意：服务停止时的 sysstat 报告

当 `sysstat` 服务停止时（例如，在重引导或关机期间），该工具仍会通过自动运行 **/usr/lib64/sa/sa1 -S ALL 1 1** 命令收集最后一分钟的统计。收集的二进制数据存储在系统活动数据文件中。

2.1.3.1.1 CPU 使用率报告：sar

如果不结合使用任何选项调用 **sar**，它会显示有关 CPU 使用率的基本报告。在多处理器计算机上，将汇总所有 CPU 的结果。使用选项 `-P ALL` 还可查看各个 CPU 的统计数据。

```
# sar 10 5
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_
  (2 CPU)

17:51:29        CPU     %user     %nice   %system   %iowait    %steal     %idle
17:51:39        all     57,93      0,00      9,58      1,01      0,00     31,47
17:51:49        all     32,71      0,00      3,79      0,05      0,00     63,45
17:51:59        all     47,23      0,00      3,66      0,00      0,00     49,11
17:52:09        all     53,33      0,00      4,88      0,05      0,00     41,74
17:52:19        all     56,98      0,00      5,65      0,10      0,00     37,27
```

```
Average:        all     49,62      0,00      5,51      0,24      0,00     44,62
```

%iowait 显示 CPU 在等待 I/O 请求时处于空闲状态的时间百分比。如果此值在很长一段时间内都远大于零，则表示 I/O 系统（网络或硬盘）存在瓶颈。如果 %idle 值在很长一段时间内都为零，表示您的 CPU 正在满负荷工作。

2.1.3.1.2 内存使用率报告：sar -r

使用选项 -r 生成系统内存 (RAM) 的总体情况说明。

```
# sar -r 10 5
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_        (2 CPU)

17:55:27 kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit kbactive kbinact kbdirty
17:55:37    104232   1834624    94.62        20   627340  2677656   66.24   802052  828024    1744
17:55:47     98584   1840272    94.92        20   624536  2693936   66.65   808872  826932    2012
17:55:57     87088   1851768    95.51        20   605288  2706392   66.95   827260  821304    1588
17:56:07     86268   1852588    95.55        20   599240  2739224   67.77   829764  820888    3036
17:56:17    104260   1834596    94.62        20   599864  2730688   67.56   811284  821584    3164
Average:     96086   1842770    95.04        20   611254  2709579   67.03   815846  823746    2309
```

kbcommit 和 %commit 列显示当前工作负载可能需要的最大内存量（RAM 和交换空间）的近似值。kbcommit 显示以 KB 为单位的绝对数字，%commit 则显示百分比。

2.1.3.1.3 分页统计报告：sar -B

使用选项 -B 可显示内核分页统计数据。

```
# sar -B 10 5
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_        (2 CPU)

18:23:01 pgpgin/s pgpgout/s fault/s majflt/s pgfree/s pgscank/s pgscand/s pgsteal/s %vmeff
18:23:11   366.80     11.60  542.50     1.10  4354.80      0.00      0.00      0.00   0.00
18:23:21     0.00    333.30 1522.40     0.00 18132.40      0.00      0.00      0.00   0.00
18:23:31    47.20    127.40 1048.30     0.10 11887.30      0.00      0.00      0.00   0.00
18:23:41    46.40      2.50  336.10     0.10  7945.00      0.00      0.00      0.00   0.00
18:23:51     0.00    583.70 2037.20     0.00 17731.90      0.00      0.00      0.00   0.00
Average:    92.08    211.70 1097.30     0.26 12010.28      0.00      0.00      0.00   0.00
```

majflt/s（每秒重大错误数）列显示已从磁盘加载到内存的页数。错误可能因文件访问或错误导致。有时出现许多重大错误是正常的。例如，在应用程序启动期间就可能会出现这种情况。如果在应用程序的整个有效期内都出现重大错误，可能表示主内存不足，尤其是同时还要执行大量的直接扫描 (pgscand/s) 时。

%vmeff 列显示扫描的页数 (pgscand/s)，相对于从主内存缓存或交换缓存 (pgsteal/s) 中重复使用的页数。此值用于测量页回收的效率。正常值接近 100（正在重复使用已换出的每个非活动页）或 0（未扫描任何页）。该值不应降到 30 以下。

注意：关于现代 Linux 内核中 %vmeff 值超过 100% 的理解与处理

当 %vmeff 值超过 100% 时，表示在内存回收过程中，报告的回收页数 (pgsteal) 高于扫描页数 (pgscank)。这一现象不符合预期，因为在正常情况下，内核回收的页数不应超过其扫描的页数。

在现代 Linux 内核（5.x 及以上版本）中，内存管理和页面回收统计方式的更改使得 %vmeff 作为虚拟内存效率指标的可靠性降低。这些更改包括：将后台回收与直接回收的指标分离、以更细粒度跟踪内存操作、引入延迟回收机制，以及增加对 NUMA 感知和基于 cgroup 的内存管理的支持。此类更新可能导致 **sar** 等工具误读 pgsteal 和 pgscank 指标，致使某些情况下出现 %vmeff 值超过 100% 的现象。

如果 %vmeff 超过 100%，请不要将其作为性能指标。建议改为通过以下方式分析内存性能：监控 pgpgin/s 和 pgpgout/s 了解分页活动，监控 majflt/s 和 fault/s 了解缺页情况，以及监控 `/proc/meminfo` 中的内存使用详情（如 Active、Inactive、Dirty 和 Writeback）。此外，应将上述指标与应用程序级性能及 I/O 模式关联分析，以识别潜在的内存瓶颈或低效问题。

2.1.3.1.4 块设备统计报告：sar -d

使用选项 -d 可显示块设备（硬盘、光盘驱动器、USB 存储设备等）。请务必使用附加选项 -p（美观打印），使 DEV 列更易阅读。

```
# sar -d -p 10 5
 Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_        (2 CPU)

18:46:09 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18:46:19 sda  1.70    33.60      0.00     19.76      0.00      0.47      0.47      0.08
18:46:19 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18:46:19 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18:46:29 sda  8.60   114.40    518.10     73.55      0.06      7.12      0.93      0.80
18:46:29 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

```
18:46:29 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18:46:39 sda 40.50  3800.80    454.90    105.08      0.36      8.86      0.69      2.80
18:46:39 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18:46:39 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18:46:49 sda  1.40     0.00    204.90    146.36      0.00      0.29      0.29      0.04
18:46:49 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18:46:49 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18:46:59 sda  3.30     0.00    503.80    152.67      0.03      8.12      1.70      0.56
18:46:59 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

Average: DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
Average: sda 11.10   789.76    336.34    101.45      0.09      8.07      0.77      0.86
Average: sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

比较所有磁盘的 tps、rd_sec/s 和 wr_sec/s 的 Average 值。如果 svctm 和 %util 列中的值一直较大，可能表示 I/O 子系统是瓶颈。

如果计算机使用多个磁盘，则最好是以交错方式在等速、等容量的磁盘之间均匀分布 I/O。需要考虑到存储系统是否有多层。此外，如果有多个存储路径，请在平衡存储空间的使用方式时，考虑链接饱和度的范围应为多少。

2.1.3.1.5 网络统计报告：`sar -n KEYWORD`

选项 `-n` 可让您生成多份网络相关报告。与 `-n` 一起指定以下关键字之一：

- **DEV**：生成所有网络设备的统计报告
- **EDEV**：生成所有网络设备的错误统计报告
- **NFS**：生成 NFS 客户端的统计报告
- **NFSD**：生成 NFS 服务器的统计报告
- **SOCK**：生成有关套接字的统计报告
- **ALL**：生成所有网络统计报告

2.1.3.2 可视化 `sar` 数据

`sar` 报告对于用户来说有时并不容易解读。而 kSar 这款 Java 应用程序能将 **`sar`** 数据可视化，生成直观的图表。它甚至还可以生成 PDF 报告。kSar 可接受实时生成的数据以及文件中的既往数据。kSar 通过 BSD 许可证授权，可从 https://sourceforge.net/projects/ksar/ 获取。

2.2 系统信息

2.2.1 设备负载信息：iostat

要监控系统设备负载，请使用 **iostat**。此工具生成的报告可用于在挂接到系统的物理磁盘之间更好地进行负载平衡。

要使用 **iostat**，请安装软件包 sysstat。

第一份 **iostat** 报告显示自引导系统以来收集的统计信息。后续报告则涵盖上一份报告之后的时间。

```
> iostat
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_
  (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17.68    4.49    4.24    0.29    0.00   73.31

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sdb               2.02        36.74        45.73    3544894    4412392
sda               1.05         5.12        13.47     493753    1300276
sdc               0.02         0.14         0.00      13641         37
```

以这种方式调用 **iostat** 可帮助您确定吞吐量是否与预期不符，但不会解释原因。通过调用 **iostat** `-x` 生成的扩展报告可以更好地解答此类问题。例如，扩展报告中额外包含有关平均队列大小和平均等待时间的信息。如果使用 `-z` 开关排除空闲块设备，还可以更轻松地评估数据。在 **iostat** 的手册页 (**man 1 iostat**) 中可以找到显示的每个列标题的定义。

您还可以指定应按指定的间隔监控的特定设备。例如，要按三秒间隔为设备 sda 生成五份报告，请使用：

```
> iostat -p sda 3 5
```

要显示网络文件系统 (NFS) 的统计数据，可以使用下列两个类似的实用程序：

- **nfsiostat-sysstat** 已随附于软件包 sysstat 中。
- **nfsiostat** 已随附于软件包 nfs-client 中。

注意：在多路径设置中使用 iostat

iostat 命令可能不会显示 **nvme list-subsys** 列出的所有控制器。默认情况下，**iostat** 会过滤掉所有没有 I/O 的块设备。要使 **iostat** 显示**所有**设备，请使用以下命令：

```
> iostat -p ALL
```

2.2.2 处理器活动监控：mpstat

实用程序 **mpstat** 检查每个可用处理器的活动。如果您的系统仅有一个处理器，将报告全局平均统计数据。

计时参数的工作方式与 **iostat** 命令相同。输入 **mpstat 2 5** 会以两秒间隔列显所有处理器的五份报告。

```
# mpstat 2 5
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_        (2 CPU)

13:51:10  CPU   %usr  %nice  %sys  %iowait  %irq  %soft  %steal  %guest  %gnice   %idle
13:51:12  all   8,27   0,00  0,50     0,00  0,00   0,00    0,00    0,00    0,00   91,23
13:51:14  all  46,62   0,00  3,01     0,00  0,00   0,25    0,00    0,00    0,00   50,13
13:51:16  all  54,71   0,00  3,82     0,00  0,00   0,51    0,00    0,00    0,00   40,97
13:51:18  all  78,77   0,00  5,12     0,00  0,00   0,77    0,00    0,00    0,00   15,35
13:51:20  all  51,65   0,00  4,30     0,00  0,00   0,51    0,00    0,00    0,00   43,54
Average:  all  47,85   0,00  3,34     0,00  0,00   0,40    0,00    0,00    0,00   48,41
```

从 **mpstat** 数据中可以看到：

- %usr 与 %sys 之比。例如，比值 10:1 表示工作负载主要是运行应用程序代码，分析重点应该是应用程序。比值 1:10 表示工作负载主要在内核方面受到限制，值得考虑微调内核。或者，确定应用程序为何在内核方面受到限制，以及是否可以缓解这种情况。
- 即使在系统负载总体较轻的情况下，是否仍有一部分 CPU 几乎被完全使用。少量热点 CPU 可能表示工作负载未并行化，在处理器更少但更快的计算机上执行可能更有利。

2.2.3 处理器频率监控：turbostat

turbostat 显示 AMD64/Intel 64 处理器的频率、负载、温度和算力。此工具能够以两种模式运行：如果使用某个命令调用，将派生命令进程，且命令完成时将显示统计数据。如果不使用命令运行，它每隔五秒会显示一次更新的统计数据。使用 **turbostat** 需要加载内核模块 `msr`。

```
> sudo turbostat find /etc -type d -exec true {} \;
0.546880 sec
     CPU Avg_MHz   Busy% Bzy_MHz TSC_MHz
       -     416   28.43    1465    3215
       0     631   37.29    1691    3215
       1     416   27.14    1534    3215
       2     270   24.30    1113    3215
       3     406   26.57    1530    3214
       4     505   32.46    1556    3214
       5     270   22.79    1184    3214
```

输出可能会因 CPU 类型而异。要显示温度和算力等更多细节，请使用 `--debug` 选项。有关更多命令行选项以及字段说明的解释，请参见 **man 8 turbostat**。

2.2.4 任务监控：pidstat

如果您需要查看特定任务对您系统施加的负载，请使用 **pidstat** 命令。此命令会列显每个选定任务的活动，或列显 Linux 内核管理的所有任务的活动（如果未指定任务）。您还可以设置要显示的报告数，以及时间间隔。

例如，**pidstat** `-C` firefox 2 3 会列显命令名称包含字符串“firefox”的任务的负载统计数据。每隔两秒会列显三份报告。

```
# pidstat -C firefox 2 3
Linux 6.4.0-150700.38-default (jupiter)         03/11/2024        _x86_64_
  (2 CPU)

14:09:11      UID       PID    %usr %system  %guest    %CPU   CPU  Command
14:09:13     1000       387   22,77    0,99    0,00   23,76     1  firefox

14:09:13      UID       PID    %usr %system  %guest    %CPU   CPU  Command
14:09:15     1000       387   46,50    3,00    0,00   49,50     1  firefox
```

```
14:09:15      UID       PID    %usr %system  %guest    %CPU   CPU  Command
14:09:17     1000       387   60,50    7,00    0,00   67,50     1  firefox

Average:      UID       PID    %usr %system  %guest    %CPU   CPU  Command
Average:     1000       387   43,19    3,65    0,00   46,84     -  firefox
```

类似地，可以使用 **pidstat** `-d` 来评估任务正在执行多少 I/O、任务在执行这些 I/O 时是否处于休眠状态，以及任务停滞的时钟节拍数。

2.2.5 内核环形缓冲区：dmesg

Linux 内核在环形缓冲区中保存某些消息。要查看这些消息，请输入命令 **dmesg -T**。

较旧事件将记录在 `systemd` 日记中。有关日记的详细信息，请参见《管理指南》, 第 21 章 “**journalctl**：查询 systemd 日记”。

2.2.6 打开的文件的列表：lsof

要查看为具有进程 ID `PID` 的进程打开的所有文件的列表，请使用 `-p`。例如，要查看当前 shell 使用的所有文件，请输入：

```
# lsof -p $$
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
bash    8842 root  cwd    DIR   0,32      222  6772 /root
bash    8842 root  rtd    DIR   0,32      166   256 /
bash    8842 root  txt    REG   0,32   656584 31066 /bin/bash
bash    8842 root  mem    REG   0,32  1978832 22993 /lib64/libc-2.19.so
[...]
bash    8842 root    2u   CHR  136,2      0t0     5 /dev/pts/2
bash    8842 root  255u   CHR  136,2      0t0     5 /dev/pts/2
```

使用了特殊外壳变量 `$$`，它的值是外壳的进程 ID。

与 `-i` 结合使用时，**lsof** 还会列出当前打开的互联网文件：

```
# lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
wickedd-d  917 root    8u  IPv4  16627      0t0  UDP *:bootpc
```

```
wickedd-d  918 root    8u  IPv6  20752      0t0  UDP [fe80::5054:ff:fe72:5ead]:dhcpv6-client
sshd      3152 root    3u  IPv4  18618      0t0  TCP *:ssh (LISTEN)
sshd      3152 root    4u  IPv6  18620      0t0  TCP *:ssh (LISTEN)
master    4746 root   13u  IPv4  20588      0t0  TCP localhost:smtp (LISTEN)
master    4746 root   14u  IPv6  20589      0t0  TCP localhost:smtp (LISTEN)
sshd      8837 root    5u  IPv4 293709      0t0  TCP jupiter.suse.de:ssh->venus.suse.de:33619 (ESTABLISHED)
sshd      8837 root    9u  IPv6 294830      0t0  TCP localhost:x11 (LISTEN)
sshd      8837 root   10u  IPv4 294831      0t0  TCP localhost:x11 (LISTEN)
```

2.2.7 内核和 udev 事件序列查看器：`udevadm monitor`

`udevadm monitor` 侦听内核 uevent 以及 udev 规则发出的事件，并将事件的设备路径 (DEVPATH) 列显到控制台。当连接 USB 记忆棒时会出现一系列事件：

注意：监控 udev 事件

只允许 root 用户通过运行 **`udevadm`** 命令监控 udev 事件。

```
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806687] add@/class/scsi_host/host4
UEVENT[1138806687] add@/class/usb_device/usbdev4.10
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806687] add@/class/scsi_host/host4
UDEV  [1138806687] add@/class/usb_device/usbdev4.10
UEVENT[1138806692] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806692] add@/block/sdb
UEVENT[1138806692] add@/class/scsi_generic/sg1
UEVENT[1138806692] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806693] add@/class/scsi_generic/sg1
UDEV  [1138806693] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/block/sdb
UEVENT[1138806694] add@/block/sdb/sdb1
UDEV  [1138806694] add@/block/sdb/sdb1
UEVENT[1138806694] mount@/block/sdb/sdb1
UEVENT[1138806697] umount@/block/sdb/sdb1
```

2.3 流程

2.3.1 进程间通讯：ipcs

命令 **ipcs** 生成当前正在使用的 IPC 资源的列表：

```
# ipcs
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 65536      tux        600        524288     2          dest
0x00000000 98305      tux        600        4194304    2          dest
0x00000000 884738     root       600        524288     2          dest
0x00000000 786435     tux        600        4194304    2          dest
0x00000000 12058628   tux        600        524288     2          dest
0x00000000 917509     root       600        524288     2          dest
0x00000000 12353542   tux        600        196608     2          dest
0x00000000 12451847   tux        600        524288     2          dest
0x00000000 11567114   root       600        262144     1          dest
0x00000000 10911763   tux        600        2097152    2          dest
0x00000000 11665429   root       600        2336768    2          dest
0x00000000 11698198   root       600        196608     2          dest
0x00000000 11730967   root       600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xa12e0919 32768      tux        666        2
```

2.3.2 进程列表：ps

命令 **ps** 生成进程的列表。书写大多数参数时一定不能带减号。请参见 **ps --help** 获取简要帮助，或者参见手册页获取详细帮助。

要列出所有进程以及用户和命令行信息，请使用 **ps axu**：

```
> ps axu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  34376  4608 ?        Ss   Jul24   0:02 /usr/lib/systemd/systemd
root         2  0.0  0.0      0     0 ?        S    Jul24   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Jul24   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Jul24   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    Jul24   0:00 [kworker/u2:0]
root         7  0.0  0.0      0     0 ?        S    Jul24   0:00 [migration/0]
[...]
tux      12583  0.0  0.1 185980  2720 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-mtp-volume-monitor
tux      12587  0.0  0.1 198132  3044 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
tux      12591  0.0  0.1 181940  2700 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-goa-volume-monitor
tux      12594  8.1 10.6 1418216 163564 ?      Sl   10:12   0:03 /usr/bin/gnome-shell
tux      12600  0.0  0.3 393448  5972 ?        Sl   10:12   0:00 /usr/lib/gnome-settings-daemon-3.0/gsd-
printer
tux      12625  0.0  0.6 227776 10112 ?        Sl   10:12   0:00 /usr/lib/gnome-control-center-search-
provider
tux      12626  0.5  1.5 890972 23540 ?        Sl   10:12   0:00 /usr/bin/nautilus --no-default-window
[...]
```

要检查有多少个 **sshd** 进程正在运行，请将选项 `-p` 与命令 **pidof** 一起使用，这会列出给定进程的进程 ID。

```
> ps -p $(pidof sshd)
  PID TTY      STAT   TIME COMMAND
 1545 ?        Ss     0:00 /usr/sbin/sshd -D
 4608 ?        Ss     0:00 sshd: root@pts/0
```

可以根据需要设置进程列表的格式。选项 `L` 返回所有关键字的列表。输入以下命令可以生成所有进程按内存使用量排序的列表：

```
> ps ax --format pid,rss,cmd --sort rss
  PID   RSS CMD
  PID   RSS CMD
    2     0 [kthreadd]
    3     0 [ksoftirqd/0]
    4     0 [kworker/0:0]
    5     0 [kworker/0:0H]
    6     0 [kworker/u2:0]
    7     0 [migration/0]
    8     0 [rcu_bh]
[...]
12518 22996 /usr/lib/gnome-settings-daemon-3.0/gnome-settings-daemon
12626 23540 /usr/bin/nautilus --no-default-window
12305 32188 /usr/bin/Xorg :0 -background none -verbose
```

```
12594 164900 /usr/bin/gnome-shell
```

有用的 ps 调用

ps aux--sort COLUMN

按 COLUMN 对输出进行排序。请将 COLUMN 替换为

pmem（表示物理内存比率）

pcpu（表示 CPU 比率）

rss（表示常驻集大小，即不交换的物理内存）

ps axo pid,%cpu,rss,vsz,args,wchan

显示每个进程、其 PID、CPU 用量、内存大小（常驻和虚拟）、名称及其系统调用。

ps axfo pid,args

显示进程树。

2.3.3 进程树：pstree

命令 **pstree** 生成树状进程列表：

```
> pstree
systemd---accounts-daemon---{gdbus}
        |                 |-{gmain}
        |-at-spi-bus-laun---dbus-daemon
        |                 |-{dconf worker}
        |                 |-{gdbus}
        |                 |-{gmain}
        |-at-spi2-registr---{gdbus}
        |-cron
        |-2*[dbus-daemon]
        |-dbus-launch
        |-dconf-service---{gdbus}
        |               |-{gmain}
        |-gconfd-2
        |-gdm---gdm-simple-slav---Xorg
        |     |                 |-gdm-session-wor---gnome-session---gnome-setti+
        |     |                 |                 |               |-gnome-shell+
++
```

```
        |     |                 |                 |               |-{dconf work+
        |     |                 |                 |               |-{gdbus}
        |     |                 |                 |               |-{gmain}
        |     |                 |                 |-{gdbus}
        |     |                 |                 |-{gmain}
        |     |                 |-{gdbus}
        |     |                 |-{gmain}
        |     |-{gdbus}
        |     |-{gmain}
[...]
```

参数 `-p` 将进程 ID 添加到给定的名称。要让命令行也显示出来，请使用 `-a` 参数：

2.3.4 进程表：top

命令 **`top`**（“table of processes”（进程表）的英文缩写）会显示一个进程列表，该列表每两秒自动刷新一次。要停止该程序，请按 `Q` 键。参数 `-n 1` 可在显示一次进程列表后停止该程序。下面是 **`top -n 1`** 命令的示例输出：

```
> top -n 1
Tasks: 128 total,   1 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.4 us,  1.2 sy,  0.0 ni, 96.3 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1535508 total,   699948 used,   835560 free,      880 buffers
KiB Swap:  1541116 total,        0 used,  1541116 free.   377000 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0  116292   4660   2028 S 0.000 0.303   0:04.45 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:00.07 ksoftirqd+
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0+
    6 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kworker/u+
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.00 migration+
    8 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S 0.000 0.000   0:00.24 rcu_sched
   10 root      rt   0       0      0      0 S 0.000 0.000   0:00.01 watchdog/0
   11 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 khelper
   12 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kdevtmpfs
   13 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 netns
```

```
   14 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 writeback
   15 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kintegrit+
   16 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 bioset
   17 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 crypto
   18 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kblockd
```

默认情况下，输出将按 CPU 用量（列 %CPU，快捷方式为 `Shift`–`P`）排序。使用以下组合键可以更改排序字段：

`Shift`–`M`：常驻内存 (RES)

`Shift`–`N`：进程 ID (PID)

`Shift`–`T`：时间 (TIME+)

要使用任何其他字段进行排序，请按 `F` 并从列表中选择一个字段。要切换排序顺序，请使用 `Shift`–`R`。

参数 `-U UID` 只监控与特定用户关联的进程。请将 `UID` 替换为用户的用户 ID。使用 **`top -U $(id -u)`** 显示当前用户的进程

2.3.5 IBM Z 超级管理程序监控器：**hyptop**

hyptop 通过 debugfs 使用内核基础架构来提供 IBM Z 超级管理程序环境的动态实时视图。它适用于 z/VM 或 LPAR 超级管理程序。例如，它可以根据可用的数据，显示活动 LPAR 或 z/VM Guest 的 CPU 与内存消耗量。它提供类似于 **top** 命令的基于 curses 的用户界面。**hyptop** 提供两个窗口：

- sys_list：列出当前超级管理程序正在运行的系统
- sys：更详细地显示一个系统

您可以在交互模式（默认）下或使用 `-b` 选项在批模式下运行 **hyptop**。在交互模式下，启动 **hyptop** 后，可以按 `?` 获取帮助。

LPAR 下的 sys_list 窗口输出：

```
12:30:48 | CPU-T: IFL(18) CP(3) UN(3)     ?=help
system  #cpu    cpu   mgm    Cpu+  Mgm+   online
(str)    (#)    (%)   (%)    (hm)  (hm)    (dhm)
H05LP30   10 461.14 10.18 1547:41  8:15 11:05:59
```

```
H05LP33    4 133.73  7.57  220:53  6:12 11:05:54
H05LP50    4  99.26  0.01  146:24  0:12 10:04:24
H05LP02    1  99.09  0.00  269:57  0:00 11:05:58
TRX2CFA    1   2.14  0.03    3:24  0:04 11:06:01
H05LP13    6   1.36  0.34    4:23  0:54 11:05:56
TRX1      19   1.22  0.14   13:57  0:22 11:06:01
TRX2      20   1.16  0.11   26:05  0:25 11:06:00
H05LP55    2   0.00  0.00    0:22  0:00 11:05:52
H05LP56    3   0.00  0.00    0:00  0:00 11:05:52
         413 823.39 23.86 3159:57 38:08 11:06:01
```

z/VM 下的“sys_list”窗口输出：

```
12:32:21 | CPU-T: UN(16)                          ?=help
system   #cpu    cpu    Cpu+   online memuse memmax wcur
(str)     (#)    (%)    (hm)    (dhm)  (GiB)  (GiB)  (#)
T6360004    6 100.31  959:47 53:05:20   1.56   2.00  100
T6360005    2   0.44    1:11  3:02:26   0.42   0.50  100
T6360014    2   0.27    0:45 10:18:41   0.54   0.75  100
DTCVSW1     1   0.00    0:00 53:16:42   0.01   0.03  100
T6360002    6   0.00  166:26 40:19:18   1.87   2.00  100
OPERATOR    1   0.00    0:00 53:16:42   0.00   0.03  100
T6360008    2   0.00    0:37 30:22:55   0.32   0.75  100
T6360003    6   0.00 3700:57 53:03:09   4.00   4.00  100
NSLCF1      1   0.00    0:02 53:16:41   0.03   0.25  500
EREP        1   0.00    0:00 53:16:42   0.00   0.03  100
PERFSVM     1   0.00    0:53  2:21:12   0.04   0.06    0
TCPIP       1   0.00    0:01 53:16:42   0.01   0.12 3000
DATAMOVE    1   0.00    0:05 53:16:42   0.00   0.03  100
DIRMAINT    1   0.00    0:04 53:16:42   0.01   0.03  100
DTCVSW2     1   0.00    0:00 53:16:42   0.01   0.03  100
RACFVM      1   0.00    0:00 53:16:42   0.01   0.02  100
           75 101.57 5239:47 53:16:42  15.46  22.50 3000
```

LPAR 下的 sys 窗口输出：

```
14:08:41 | H05LP30 | CPU-T: IFL(18) CP(3) UN(3)                 ? = help
cpuid   type    cpu   mgm visual.
(#)    (str)    (%)   (%) (vis)
0        IFL  96.91  1.96 |############################################ |
```

```
1        IFL  81.82  1.46 |##################################        |
2        IFL  88.00  2.43 |#####################################     |
3        IFL  92.27  1.29 |#######################################   |
4        IFL  83.32  1.05 |###################################       |
5        IFL  92.46  2.59 |#######################################   |
6        IFL   0.00  0.00 |                                          |
7        IFL   0.00  0.00 |                                          |
8        IFL   0.00  0.00 |                                          |
9        IFL   0.00  0.00 |                                          |
             534.79 10.78
```

z/VM 下的 sys 窗口输出：

```
15:46:57 | T6360003 | CPU-T: UN(16)                  ? = help
cpuid     cpu visual
(#)       (%) (vis)
0      548.72 |#######################################    |
        548.72
```

2.3.6 顶层式 I/O 监控器：iotop

iotop 实用程序按进程或线程显示 I/O 使用情况表。

注意：安装 iotop

默认未安装 **iotop**。您需要以 `root` 身份使用 **zypper in iotop** 手动安装它。

iotop 显示采样期间每个进程读取和写入的 I/O 带宽的列。它还会显示该进程在换入时以及等待 I/O 时所花时间的百分比。对于每个进程，都会显示 I/O 优先级（类/级别）。此外，还会在界面顶部显示采样期间读取和写入的总 I/O 带宽。

- 使用 `←` 和 `→` 键可更改排序。
- 使用 `R` 键可反转排序顺序。
- 使用 `O` 键可在显示所有进程和线程（默认视图）与仅显示执行 I/O 的进程和线程之间切换。（此功能类似于在命令行上添加 `--only`。）

- 使用 `P` 键可在显示线程（默认视图）与显示进程之间切换。（此功能类似于 `--only`。）
- 使用 `A` 键可在显示当前 I/O 带宽（默认视图）与显示自 **iotop** 启动以来的累计 I/O 操作数目之间切换。（此功能类似于 `--accumulated`。）
- 使用 `I` 键可更改某个线程或者某个进程的线程的优先级。
- 按 `Q` 键会退出 **iotop**。
- 按任何其他键会强制刷新。

下面是当 **find** 和 **emacs** 正在运行时命令 **iotop --only** 的示例输出：

```
# iotop --only
Total DISK READ: 50.61 K/s | Total DISK WRITE: 11.68 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 3416 be/4 tux         50.61 K/s    0.00 B/s  0.00 %  4.05 % find /
  275 be/3 root        0.00 B/s    3.89 K/s  0.00 %  2.34 % [jbd2/sda2-8]
 5055 be/4 tux          0.00 B/s    3.89 K/s  0.00 %  0.04 % emacs
```

还可以在批模式 (`-b`) 下使用 **iotop**，其输出将存储在文件中以供日后分析。有关完整的选项集，请参见手册页 (**man 8 iotop**)。

2.3.7 修改进程的资源优先级：nice 和 renice

内核按照进程的 nice 级别（也称为资源优先级）决定哪些进程需要的 CPU 时间比其他进程更多。进程的“nice”级别越高，它占用的 CPU 时间较之其他进程就越少。nice 级别的范围从 -20（最低 “nice” 级别）到 19。负值只能由 `root` 用户设置。

当运行持续时间较长且会使用大量 CPU 时间的非时间关键型进程时，调整资源优先级会很有用。例如，在同时执行其他任务的系统上编译内核时。将此类进程的“nice”级别调高可确保其他任务（例如 Web 服务器）获得更高的优先级。

调用不带任何参数的 **nice** 会列显当前资源优先级：

```
> nice
0
```

运行 **nice** COMMAND 会将给定命令的当前 nice 级别加 10。使用 **nice** -n LEVEL COMMAND 可以指定相对于当前级别的新资源优先级。

要更改正在运行的进程的资源优先级，请使用 **renice** PRIORITY -p PROCESS_ID，例如：

```
> renice +5 3266
```

要恢复特定用户拥有的所有进程的资源优先级，请使用选项 -u USER。 使用选项 -g PROCESS_GROUP_ID 可重新设置进程组的资源优先级。

2.4 内存

2.4.1 内存用量：free

实用程序 **free** 可检查 RAM 和交换空间用量。将显示有关可用内存和已使用内存以及交换区域的详细信息：

```
> free
             total       used       free     shared    buffers     cached
Mem:      32900500   32703448     197052          0     255668    5787364
-/+ buffers/cache:   26660416    6240084
Swap:      2046972     304680    1742292
```

选项 -b、-k、-m、-g 分别以字节、KB、MB 或 GB 为单位显示输出。参数 -s delay 确保显示内容每隔 DELAY 秒刷新。例如，**free -s 1.5** 每隔 1.5 秒生成更新内容。

2.4.2 内存使用详情：/proc/meminfo

相比 **free**，使用 /proc/meminfo 可以获取有关内存使用情况的更详细信息。实际上，**free** 会使用此文件中一定数量的数据。下面是在 64 位系统上的示例输出。由于内存管理方式不同，在 32 位系统上的输出略有不同：

```
MemTotal:        1942636 kB
MemFree:         1294352 kB
MemAvailable:    1458744 kB
Buffers:             876 kB
```

```
Cached:           278476 kB
SwapCached:            0 kB
Active:           368328 kB
Inactive:         199368 kB
Active(anon):     288968 kB
Inactive(anon):    10568 kB
Active(file):      79360 kB
Inactive(file):   188800 kB
Unevictable:          80 kB
Mlocked:              80 kB
SwapTotal:       2103292 kB
SwapFree:        2103292 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        288592 kB
Mapped:            70444 kB
Shmem:             11192 kB
Slab:              40916 kB
SReclaimable:      17712 kB
SUnreclaim:        23204 kB
KernelStack:        2000 kB
PageTables:        10996 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     3074608 kB
Committed_AS:    1407208 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       145996 kB
VmallocChunk:   34359588844 kB
HardwareCorrupted:     0 kB
AnonHugePages:     86016 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       79744 kB
```

```
DirectMap2M:     2017280 kB
```

这些项的含义如下：

MemTotal

RAM 总量。

MemFree

未使用的 RAM 量。

MemAvailable

在不交换的情况下可用于启动新应用程序的预计内存量。

Buffers

RAM 中包含文件系统元数据的文件缓冲区缓存。

Cached

RAM 中的页缓存。这不包括缓冲区缓存和交换缓存，但包括 Shmem 内存。

SwapCached

换出内存的页缓存。

Active、Active(anon)、Active(file)

最近使用的内存，除非必要或明确请求，否则不会回收。Active 是 Active(anon) 与 Active(file) 之和。

- Active(anon) 跟踪交换支持的内存。这包括专用和共享的匿名映射，以及完成写入时复制操作后的专用文件页。
- Active(file) 跟踪其他文件系统支持的内存。

Inactive、Inactive(anon)、Inactive(file)

上次使用时间较早的内存，通常会最先回收。Inactive 是 Inactive(anon) 与 Inactive(file) 之和。

- Inactive(anon) 跟踪交换支持的内存。这包括专用和共享的匿名映射，以及完成写入时复制操作后的专用文件页。
- Inactive(file) 跟踪其他文件系统支持的内存。

Unevictable

无法回收的内存量（例如，由于它是 Mlocked 或用作 RAM 磁盘）。

Mlocked

由 `mlock` 系统调用提供支持的内存量。`mlock` 允许进程定义要将其虚拟内存映射到物理 RAM 的哪个部分。但是，`mlock` 不保证实现这种定位。

SwapTotal

交换空间量。

SwapFree

未使用的交换空间量。

Dirty

由于包含更改而正在等待写入磁盘的内存量（相较于后备存储空间）。脏数据可在短暂延迟后由应用程序或内核明确同步。将大量脏数据写入磁盘可能需要花费大量时间，从而导致停滞。可使用 **`sysctl`** 参数 `vm.dirty_ratio` 或 `vm.dirty_bytes` 来控制可在任意时间存在的脏数据总量（有关更多细节，请参见第 15.1.5 节 “写回”）。

Writeback

当前正在写入磁盘的内存量。

Mapped

使用 `mmap` 系统调用声明的内存。

Shmem

在进程组之间共享的内存，例如 IPC 数据、`tmpfs` 数据和共享的匿名内存。

Slab

内核内部数据结构的内存分配。

SReclaimable

可回收的 Slab 部分，例如缓存（inode、dentry 等）。

SUnreclaim

不可回收的 Slab 部分。

KernelStack

应用程序（通过系统调用）使用的内核空间内存量。

PageTables

专用于所有进程的页表的内存量。

NFS_Unstable

已发送到服务器，但尚未在服务器中提交的 NFS 页。

Bounce

块设备的回弹缓冲区使用的内存。

WritebackTmp

临时写回缓冲区的 FUSE 使用的内存。

CommitLimit

根据过量使用率设置提供给系统的内存量。仅当启用了严格的过量使用统计时，才会强制此限制。

Committed_AS

当前工作负载在最坏情况下需要的内存总量（RAM 和交换空间）近似值。

VmallocTotal

分配的内核虚拟地址空间量。

VmallocUsed

已使用的内核虚拟地址空间量。

VmallocChunk

可用内核虚拟地址空间的最大连续块。

HardwareCorrupted

有故障的内存量（仅当使用 ECC RAM 时才可检测到）。

AnonHugePages

映射到用户空间页表的匿名大页。这些页是在未经专门请求的情况下以透明方式为进程分配的，因此它们也称为**透明大页** (THP)。

HugePages_Total

供 SHM_HUGETLB 和 MAP_HUGETLB 使用的预分配大页数，或者根据 `/proc/sys/vm/nr_hugepages` 中的定义通过 `hugetlbfs` 文件系统使用的预分配大页数。

HugePages_Free

可用大页数。

HugePages_Rsvd

已提交的大页数。

HugePages_Surp

超出 HugePages_Total 的可用大页数（“超额页”），在 `/proc/sys/vm/nr_overcommit_hugepages` 中定义。

Hugepagesize

大页的大小 — 在 AMD64/Intel 64 上，默认值为 2048 KB。

DirectMap4k 等

映射到具有给定大小（示例中为 4 kB）的页的内核内存量。

2.4.3 进程内存用量：smaps

使用 **top** 或 **ps** 等标准工具无法确切地确定特定进程消耗的内存量。如果您需要确切的数据，请使用内核 2.6.14 中引入的 smaps 子系统。该子系统位于 `/proc/PID/smaps` 中，会显示 ID 为 `PID` 的进程当时使用的干净和脏内存页数。它会区分共享内存和专用内存，因此您可以查看进程使用的内存量，其中不包括与其他进程共享的内存。有关详细信息，请参见 `/usr/src/linux/Documentation/filesystems/proc.txt`（需要安装软件包 `kernel-source`）。

读取 smaps 会产生很高的开销。因此，不建议经常性地对其进行监控，而是仅在密切监控特定进程时才这样做。

2.4.4 numaTOP

numaTOP 是适用于 NUMA（非一致性内存访问）系统的一个工具。该工具提供 NUMA 系统的实时分析，可帮助识别 NUMA 相关的性能瓶颈。

一般而言，numaTOP 可让您通过分析远程内存访问 (RMA) 数量、本地内存访问 (LMA) 数量和 RMA/LMA 比率，来识别和调查位置不佳（即，本地内存用量与远程内存用量之比不佳）的进程和线程。

PowerPC 以及以下 Intel Xeon 处理器支持 numaTOP：5500 系列、6500/7500 系列、5600 系列、E7-x8xx 系列和 E5-16xx/24xx/26xx/46xx 系列。

numaTOP 在官方软件储存库中提供，您可以使用 **sudo zypper in numatop** 命令安装该工具。要启动 numaTOP，请运行 **numatop** 命令。要大致了解 numaTOP 功能和用法，请使用 **man numatop** 命令。

2.5 网络

提示：流量整形

如果网络带宽低于预期，您应该先检查是否对网段激活了任何流量整形规则。

2.5.1 基本网络诊断：ip

ip 是用于设置和控制网络接口的强大工具。还可以使用它来快速查看有关系统网络接口的基本统计数据。例如，接口是否已启动，或是发生了多少错误、丢包或包冲突。

如果不结合使用任何附加参数运行 **ip**，它会显示帮助输出。要列出所有网络接口，请输入 **ip addr show**（或缩写形式 **ip a**）。**ip addr show up** 仅列出正在运行的网络接口。**ip -s link show** DEVICE 仅列出指定接口的统计数据：

```
# ip -s link show br0
6: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode
 DEFAULT
    link/ether 00:19:d1:72:d4:30 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    6346104756 9265517  0       10860   0       0
    TX: bytes  packets  errors  dropped carrier collsns
    3996204683 3655523  0       0       0       0
```

ip 还可以显示接口 (link)、路由表 (route) 等信息 - 有关细节，请参见 **man 8 ip**。

```
# ip route
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth0  proto kernel  scope link  src 192.168.2.100
192.168.2.0/24 dev eth1  proto kernel  scope link  src 192.168.2.101
192.168.2.0/24 dev eth2  proto kernel  scope link  src 192.168.2.102
```

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:44:30:51 brd ff:ff:ff:ff:ff:ff
```

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:a3:c1:fb brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:32:a4:09 brd ff:ff:ff:ff:ff:ff
```

2.5.2 显示进程的网络用量：nethogs

在某些情况下（例如，如果网络流量突然变得过高），需要使用该工具来迅速找到引发高流量的应用程序。**nethogs** 工具在设计上与 **top** 类似，可显示所有相关进程的传入和传出流量：

```
PID   USER  PROGRAM                                    DEV   SENT   RECEIVED
27145 root   zypper                                    eth0  5.719  391.749 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30015             0.102    2.326 KB/sec
26635 tux    /usr/lib64/firefox/firefox                eth0  0.026    0.026 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045             0.000    0.021 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045             0.000    0.018 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30015             0.000    0.018 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045             0.000    0.017 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045             0.000    0.017 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045             0.069    0.000 KB/sec
?     root   unknown TCP                                     0.000    0.000 KB/sec

TOTAL                                                        5.916  394.192 KB/sec
```

与 **top** 一样，**nethogs** 支持交互式命令：

M：在显示模式（kb/s、kb、b、mb）之间循环

R：按 RECEIVED 排序

S：按 SENT 排序

Q：退出

2.5.3 以太网卡细节：ethtool

ethtool 可详细显示和更改以太网设备的各个方面。默认情况下，它会列显指定设备的当前设置。

```
# ethtool eth0
Settings for eth0:
 Supported ports: [ TP ]
 Supported link modes:   10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
 Supports auto-negotiation: Yes
 Advertised link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
 Advertised pause frame use: No
[...]
 Link detected: yes
```

下表显示了可用于查询设备特定信息的 **ethtool** 选项：

表 2.1： ethtool 的查询选项列表

ethtool option	查询设备的以下信息
-a	暂停参数信息
-c	中断合并信息
-g <组文件>	Rx/Tx（接收/传输）环参数信息
-i	关联的驱动程序信息
-k	减负信息
-S	NIC 和驱动程序特定的统计数据

2.5.4 显示网络状态：ss

ss 是用于转储套接字统计数据的工具，可取代 **netstat** 命令。要列出所有连接，请使用不带参数的 **ss**：

```
# ss
```

```
Netid  State      Recv-Q Send-Q   Local Address:Port       Peer Address:Port
u_str  ESTAB      0      0                    * 14082                 * 14083
u_str  ESTAB      0      0                    * 18582                 * 18583
u_str  ESTAB      0      0                    * 19449                 * 19450
u_str  ESTAB      0      0      @/tmp/dbus-gmUUwXABPV 18784           * 18783
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 19383 * 19382
u_str  ESTAB      0      0      @/tmp/dbus-gmUUwXABPV 18617           * 18616
u_str  ESTAB      0      0      @/tmp/dbus-58TPPDv8qv 19352           * 19351
u_str  ESTAB      0      0                    * 17658                 * 17657
u_str  ESTAB      0      0                    * 17693                 * 17694
[..]
```

要显示当前打开的所有网络端口，请使用以下命令：

```
# ss -l
Netid  State      Recv-Q Send-Q      Local Address:Port  Peer Address:Port
nl     UNCONN     0      0                 rtnl:4195117                *
nl     UNCONN     0      0       rtnl:wickedd-auto4/811                *
nl     UNCONN     0      0       rtnl:wickedd-dhcp4/813                *
nl     UNCONN     0      0                 rtnl:4195121                *
nl     UNCONN     0      0                 rtnl:4195115                *
nl     UNCONN     0      0       rtnl:wickedd-dhcp6/814                *
nl     UNCONN     0      0                  rtnl:kernel                *
nl     UNCONN     0      0             rtnl:wickedd/817                *
nl     UNCONN     0      0                 rtnl:4195118                *
nl     UNCONN     0      0                rtnl:nscd/706                *
nl     UNCONN     4352   0              tcpdiag:ss/2381                *
[...]
```

显示网络连接时，您可以指定要显示的套接字类型，例如 TCP (`-t`) 或 UDP (`-u`)。`-p` 选项可显示套接字所属程序的 PID 和名称。

下例列出了所有 TCP 连接和使用这些连接的程序。`-a` 选项可确保显示所有已建立的连接（侦听和非侦听）。`-p` 选项可显示套接字所属程序的 PID 和名称。

```
# ss -t -a -p
State    Recv-Q Send-Q  Local Address:Port   Peer Address:Port
LISTEN   0      128                 *:ssh               *:*  users:(("sshd",1551,3))
LISTEN   0      100         127.0.0.1:smtp              *:*  users:(("master",1704,13))
ESTAB    0      132      10.120.65.198:ssh  10.120.4.150:55715  users:(("sshd",2103,5))
LISTEN   0      128                :::ssh              :::*  users:(("sshd",1551,4))
```

```
LISTEN   0      100              ::1:smtp              :::*  users:(("master",1704,14))
```

2.6 /proc 文件系统

/proc 文件系统是一个伪文件系统，在该文件系统中，内核以虚拟文件的形式保留重要信息。例如，使用以下命令显示 CPU 类型：

```
> cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 30
model name      : Intel(R) Core(TM) i5 CPU         750  @ 2.67GHz
stepping        : 5
microcode       : 0x6
cpu MHz         : 1197.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
 pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp
 lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc
 aperfmperf pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm
 sse4_1 sse4_2 popcnt lahf_lm ida dtherm tpr_shadow vnmi flexpriority ept vpid
bogomips        : 5333.85
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
[...]
```

提示：详细处理器信息

还可以运行 **x86info** 来获取有关 AMD64/Intel 64 体系结构上处理器的详细信息。

用下列命令查询中断的分配和使用：

```
> cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:        121          0          0          0   IO-APIC-edge      timer
  8:          0          0          0          1   IO-APIC-edge      rtc0
  9:          0          0          0          0   IO-APIC-fasteoi   acpi
 16:          0      11933          0          0   IO-APIC-fasteoi   ehci_hcd:+
 18:          0          0          0          0   IO-APIC-fasteoi   i801_smbus
 19:          0     117978          0          0   IO-APIC-fasteoi   ata_piix,+
 22:          0          0    3275185          0   IO-APIC-fasteoi   enp5s1
 23:     417927          0          0          0   IO-APIC-fasteoi   ehci_hcd:+
 40:    2727916          0          0          0  HPET_MSI-edge      hpet2
 41:          0    2749134          0          0  HPET_MSI-edge      hpet3
 42:          0          0    2759148          0  HPET_MSI-edge      hpet4
 43:          0          0          0    2678206  HPET_MSI-edge      hpet5
 45:          0          0          0          0   PCI-MSI-edge      aerdrv, P+
 46:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 47:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 48:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 49:          0          0          0        387   PCI-MSI-edge      snd_hda_i+
 50:     933117          0          0          0   PCI-MSI-edge      nvidia
NMI:       2102       2023       2031       1920   Non-maskable interrupts
LOC:         92         71         57         41   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:       2102       2023       2031       1920   Performance monitoring int+
IWI:      47331      45725      52464      46775   IRQ work interrupts
RTR:          2          0          0          0   APIC ICR read retries
RES:     472911     396463     339792     323820   Rescheduling interrupts
CAL:      48389      47345      54113      50478   Function call interrupts
TLB:      28410      26804      24389      26157   TLB shootdowns
TRM:          0          0          0          0   Thermal event interrupts
```

```
THR:          0          0          0          0   Threshold APIC interrupts
MCE:          0          0          0          0   Machine check exceptions
MCP:         40         40         40         40   Machine check polls
ERR:          0
MIS:          0
```

`maps` 文件中包含可执行文件和库的地址分配：

```
> cat /proc/self/maps
08048000-0804c000 r-xp 00000000 03:03 17753      /bin/cat
0804c000-0804d000 rw-p 00004000 03:03 17753      /bin/cat
0804d000-0806e000 rw-p 0804d000 00:00 0          [heap]
b7d27000-b7d5a000 r--p 00000000 03:03 11867      /usr/lib/locale/en_GB.utf8/
b7d5a000-b7e32000 r--p 00000000 03:03 11868      /usr/lib/locale/en_GB.utf8/
b7e32000-b7e33000 rw-p b7e32000 00:00 0
b7e33000-b7f45000 r-xp 00000000 03:03 8837       /lib/libc-2.3.6.so
b7f45000-b7f46000 r--p 00112000 03:03 8837       /lib/libc-2.3.6.so
b7f46000-b7f48000 rw-p 00113000 03:03 8837       /lib/libc-2.3.6.so
b7f48000-b7f4c000 rw-p b7f48000 00:00 0
b7f52000-b7f53000 r--p 00000000 03:03 11842      /usr/lib/locale/en_GB.utf8/
[...]
b7f5b000-b7f61000 r--s 00000000 03:03 9109       /usr/lib/gconv/gconv-module
b7f61000-b7f62000 r--p 00000000 03:03 9720       /usr/lib/locale/en_GB.utf8/
b7f62000-b7f76000 r-xp 00000000 03:03 8828       /lib/ld-2.3.6.so
b7f76000-b7f78000 rw-p 00013000 03:03 8828       /lib/ld-2.3.6.so
bfd61000-bfd76000 rw-p bfd61000 00:00 0          [stack]
ffffe000-fffff000 ---p 00000000 00:00 0          [vdso]
```

可以从 `/proc` 文件系统获取大量详细信息。一些重要的文件及其内容如下：

`/proc/devices`
: 可用设备

`/proc/modules`
: 加载的内核模块

`/proc/cmdline`
: 内核命令行

`/proc/meminfo`
: 有关内存使用的详细信息

`/proc/config.gz`

当前运行的内核的 **gzip** 压缩配置文件

/proc/`PID/`

`/proc/NNN` 目录中提供了当前运行进程的信息，其中 `NNN` 是相关进程的进程 ID (PID)。每个进程都可以在 `/proc/self/` 中找到其自身的特征。

文本文件 `/usr/src/linux/Documentation/filesystems/proc.txt` 中提供了更多信息（安装软件包 `kernel-source` 后会提供此文件）。

2.6.1 procinfo

命令 **procinfo** 可以汇总 `/proc` 文件系统中的重要信息：

```
> procinfo
Linux 3.11.10-17-desktop (geeko@buildhost) (gcc 4.8.1 20130909) #1 4CPU
 [jupiter.example.com]

Memory:      Total        Used        Free      Shared     Buffers      Cached
Mem:       8181908     8000632      181276           0       85472     2850872
Swap:     10481660        1576    10480084

Bootup: Mon Jul 28 09:54:13 2014    Load average: 1.61 0.85 0.74 2/904 25949

user  :       1:54:41.84  12.7%  page in :    2107312  disk 1:    52212r
 20199w
nice  :       0:00:00.46   0.0%  page out:    1714461  disk 2:    19387r
 10928w
system:       0:25:38.00   2.8%  page act:     466673  disk 3:      548r
 10w
IOwait:       0:04:16.45   0.4%  page dea:     272297
hw irq:       0:00:00.42   0.0%  page flt:  105754526
sw irq:       0:01:26.48   0.1%  swap in :          0
idle  :      12:14:43.65  81.5%  swap out:        394
guest :       0:02:18.59   0.2%
uptime:       3:45:22.24         context :   99809844
```

```
irq  0:       121 timer                irq 41:   3238224 hpet3
irq  8:         1 rtc0                 irq 42:   3251898 hpet4
irq  9:         0 acpi                 irq 43:   3156368 hpet5
irq 16:     14589 ehci_hcd:usb1        irq 45:         0 aerdrv, PCIe PME
irq 18:         0 i801_smbus           irq 46:         0 PCIe PME, pciehp
irq 19:    124861 ata_piix, ata_piix, f irq 47:         0 PCIe PME, pciehp
irq 22:   3742817 enp5s1               irq 48:         0 PCIe PME, pciehp
irq 23:    479248 ehci_hcd:usb2        irq 49:       387 snd_hda_intel
irq 40:   3216894 hpet2                irq 50:   1088673 nvidia
```

要查看所有信息，请使用参数 `-a`。参数 `-nN` 每隔 `N` 秒更新一次信息。在这种情况下，可以按 `Q` 键停止程序。

默认情况下显示累积值。使用参数 `-d` 将生成差异值。**`procinfo -dn5`** 显示最近 5 秒内更改的值：

2.6.2 系统控制参数：/proc/sys/

系统控制参数用于在运行时修改 Linux 内核参数。它们位于 `/proc/sys/` 中，可使用 **`sysctl`** 命令来查看和修改。要列出所有参数，请运行 **`sysctl`** `-a`。可以使用 **`sysctl`** `PARAMETER_NAME` 列出单个参数。

参数分为几类，可使用 **`sysctl`** `CATEGORY` 或通过列出相应目录的内容将其列出。下面列出了最重要的类别。需要安装软件包 `kernel-source` 才能获取更多阅读内容的链接。

`sysctl dev` (`/proc/sys/dev/`)

设备特定的信息。

`sysctl fs` (`/proc/sys/fs/`)

使用的文件句柄、配额和其他面向文件系统的参数。有关详细信息，请参见`/usr/src/linux/Documentation/sysctl/fs.txt`。

`sysctl kernel` (`/proc/sys/kernel/`)

有关任务调度程序、系统共享内存和其他内核相关参数的信息。有关详细信息，请参见`/usr/src/linux/Documentation/sysctl/kernel.txt`

`sysctl net` (`/proc/sys/net/`)

有关网桥和一般网络参数的信息（主要为 `ipv4/` 子目录）。有关细节，请参见 `/usr/src/linux/Documentation/sysctl/net.txt`

sysctl vm (/proc/sys/vm/)

此路径中的项与有关虚拟内存、交换和缓存的信息相关。有关细节，请参见 /usr/src/linux/Documentation/sysctl/vm.txt

要设置或更改当前会话的参数，请使用命令 **sysctl** -w PARAMETER=VALUE。要永久更改某项设置，请在 /etc/sysctl.conf 中添加 PARAMETER=VALUE 行。

2.7 硬件信息

2.7.1 PCI 资源：**lspci**

注意：访问 PCI 配置。

大多数操作系统需要 root 用户特权才能授予对计算机 PCI 配置的访问权限。

命令 **lspci** 列出 PCI 资源：

```
# lspci
00:00.0 Host bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    DRAM Controller/Host-Hub Interface (rev 01)
00:01.0 PCI bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    Host-to-AGP Bridge (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #3 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801DB/DBM \
    (ICH4/ICH4-M) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev 81)
00:1f.0 ISA bridge: Intel Corporation 82801DB/DBL (ICH4/ICH4-L) \
    LPC Interface Bridge (rev 01)
00:1f.1 IDE interface: Intel Corporation 82801DB (ICH4) IDE \
    Controller (rev 01)
```

```
00:1f.3 SMBus: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) \
    SMBus Controller (rev 01)
00:1f.5 Multimedia audio controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) AC'97 Audio Controller (rev 01)
01:00.0 VGA compatible controller: Matrox Graphics, Inc. G400/G450 (rev 85)
02:08.0 Ethernet controller: Intel Corporation 82801DB PRO/100 VE (LOM) \
    Ethernet Controller (rev 81)
```

使用 `-v` 可以生成更详细的列表：

```
# lspci -v
[...]
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet \
Controller (rev 02)
  Subsystem: Intel Corporation PRO/1000 MT Desktop Adapter
  Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 19
  Memory at f0000000 (32-bit, non-prefetchable) [size=128K]
  I/O ports at d010 [size=8]
  Capabilities: [dc] Power Management version 2
  Capabilities: [e4] PCI-X non-bridge device
  Kernel driver in use: e1000
  Kernel modules: e1000
```

从文件 `/usr/share/pci.ids` 中获取有关设备名称解析的信息。此文件中未列出的 PCI ID 标有“未知设备”。

参数 `-vv` 生成程序可查询的所有信息。要查看纯数字值，请使用参数 `-n`。

2.7.2 USB 设备：**lsusb**

命令 **lsusb** 列出所有 USB 设备。使用选项 `-v` 可列显更详细的列表。从目录 `/proc/bus/usb/` 读取详细信息。下面是在挂接集线器、内存条、硬盘和鼠标等 USB 设备情况下运行 **lsusb** 后的输出。

```
# lsusb
Bus 004 Device 007: ID 0ea0:2168 Ours Technology, Inc. Transcend JetFlash \
    2.0 / Astone USB Drive
Bus 004 Device 006: ID 04b4:6830 Cypress Semiconductor Corp. USB-2.0 IDE \
    Adapter
```

```
Bus 004 Device 005: ID 05e3:0605 Genesys Logic, Inc.
Bus 004 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 005: ID 046d:c012 Logitech, Inc. Optical Mouse
Bus 001 Device 001: ID 0000:0000
```

2.7.3 监控和微调热子系统：tmon

tmon 是能够帮助可视化、微调和测试复杂热子系统的工具。如果不使用任何参数启动，**tmon** 将以监控模式运行：

```
┌──────THERMAL ZONES(SENSORS)──────────────────────────────┐
│Thermal Zones:                 acpitz00                   │
│Trip Points:                   PC                         │
└──────────────────────────────────────────────────────────┘

┌────────────── COOLING DEVICES ───────────────────────────┐
│ID  Cooling Dev   Cur    Max   Thermal Zone Binding       │
│00    Processor     0      3   ||||||||||||               │
│01    Processor     0      3   ||||||||||||               │
│02    Processor     0      3   ||||||||||||               │
│03    Processor     0      3   ||||||||||||               │
│04 intel_powerc    -1     50   ||||||||||||               │
└──────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐
│                         10        20        30        40 │
│acpitz 0:[  8][>>>>>>>>>>P9                    C31        │
└──────────────────────────────────────────────────────────┘

┌──────────────────── CONTROLS ────────────────────────────┐
│PID gain: kp=0.36 ki=5.00 kd=0.19 Output 0.00             │
│Target Temp: 65.0C, Zone: 0, Control Device: None         │
└──────────────────────────────────────────────────────────┘


Ctrl-c - Quit   TAB - Tuning
```

有关如何解释数据、如何记录热数据，以及如何使用 **tmon** 测试和微调散热设备与传感器的详细信息，请参见手册页：**man 8 tmon**。默认情况下不会安装软件包 tmon。

2.7.4 MCELog：计算机检查异常 (MCE)

注意：可用性

此工具仅适用于 AMD64/Intel 64 系统。

发生硬件错误（包括 I/O、CPU 和内存错误）时，`mcelog` 软件包可记录和分析/转换计算机检查异常 (MCE)。此外，当发生缓存错误时，mcelog 还可以处理预测型坏页脱机和自动核心脱机。以前，此问题由每小时执行的 cron 作业管理。现在，硬件错误可由 mcelog 守护程序立即处理。

注意：AMD 可缩放 MCA 支持

SUSE Linux Enterprise Desktop 支持 AMD 的可缩放计算机检查体系结构（可缩放 MCA）。可缩放 MCA 增强了 AMD Zen 处理器中的硬件错误报告。它扩展了 MCA Bank 中记录的信息，提升了错误处理能力并提高了可诊断性。

`mcelog` 可捕获 MCA 消息（`rasdaemon` 和 `dmesg` 也可捕获 MCA 消息）。有关详细信息，请参见《Processor Programming Reference (PPR) for AMD Family 17h Model 01h, Revision B1 Processors》的第 3.1 节“Machine Check Architecture”(https://developer.amd.com/wordpress/media/2017/11/54945_PPR_Family_17h_Models_00h-0Fh.pdf)。

mcelog 在 `/etc/mcelog/mcelog.conf` 中配置。有关配置选项的信息，请参见 **`man mcelog`** 和 https://mcelog.org/。以下示例仅显示了对默认文件的更改：

```
daemon = yes
filter = yes
filter-memory-errors = yes
no-syslog = yes
logfile = /var/log/mcelog
run-credentials-user = root
run-credentials-group = nobody
client-group = root
socket-path = /var/run/mcelog-client
```

默认未启用 mcelog 服务。可以通过 YaST 系统服务编辑器或命令行启用和启动该服务：

```
# systemctl enable mcelog
# systemctl start mcelog
```

2.7.5 AMD64/Intel 64：dmidecode：DMI 表解码器

dmidecode 可显示计算机的 DMI 表，其中包含硬件的序列号和 BIOS 修订版等信息。

```
# dmidecode
# dmidecode 2.12
SMBIOS 2.5 present.
27 structures occupying 1298 bytes.
Table at 0x000EB250.

Handle 0x0000, DMI type 4, 35 bytes
Processor Information
        Socket Designation: J1PR
        Type: Central Processor
        Family: Other
        Manufacturer: Intel(R) Corporation
        ID: E5 06 01 00 FF FB EB BF
        Version: Intel(R) Core(TM) i5 CPU         750  @ 2.67GHz
        Voltage: 1.1 V
        External Clock: 133 MHz
        Max Speed: 4000 MHz
        Current Speed: 2667 MHz
        Status: Populated, Enabled
        Upgrade: Other
        L1 Cache Handle: 0x0004
        L2 Cache Handle: 0x0003
        L3 Cache Handle: 0x0001
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: Not Specified
[..]
```

2.7.6 POWER：列出硬件

lshw 可提取并显示计算机的硬件配置。

2.8 文件和文件系统

2.8.1 确定文件类型：file

命令 **file** 通过检查 /usr/share/misc/magic 来确定某个文件或文件列表的类型。

```
> file /usr/bin/file
/usr/bin/file: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), \
    for GNU/Linux 2.6.4, dynamically linked (uses shared libs), stripped
```

参数 -f LIST 指定要检查的一个或一系列文件名。-z 允许 **file** 在压缩文件的内部查找：

```
> file /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: gzip compressed data, from Unix, max compression
> file -z /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: troff or preprocessor input text \
    (gzip compressed data, from Unix, max compression)
```

参数 -i 输出 mime 类型的字符串而不是传统的说明。

```
> file -i /usr/share/misc/magic
/usr/share/misc/magic: text/plain charset=utf-8
```

2.8.2 文件系统及其用法：mount、df 和 du

命令 **mount** 会显示在哪个挂载点挂载哪个文件系统（设备和类型）：

```
# mount
/dev/sda2 on / type ext4 (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
```

```
debugfs on /sys/kernel/debug type debugfs (rw)
devtmpfs on /dev type devtmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,mode=1777)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/sda3 on /home type ext3 (rw)
securityfs on /sys/kernel/security type securityfs (rw)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
gvfs-fuse-daemon on /home/tux/.gvfs type fuse.gvfs-fuse-daemon \
(rw,nosuid,nodev,user=tux)
```

使用命令 **df** 可以获得有关文件系统全部使用情况的信息。参数 `-h`（或 `--human-readable`）将输出转换为普通用户可以理解的形式。

```
> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              20G  5,9G   13G  32% /
devtmpfs              1,6G  236K  1,6G   1% /dev
tmpfs                 1,6G  668K  1,6G   1% /dev/shm
/dev/sda3             208G   40G  159G  20% /home
```

使用命令 **du** 可以显示给定目录及其子目录中所有文件的总大小。参数 `-s` 会隐藏详细信息的输出，只提供每个参数的总计。`-h` 会再次将输出转换为直观易懂的格式：

```
> du -sh /opt
192M    /opt
```

2.8.3 有关 ELF 二进制文件的其他信息

使用 **readelf** 实用程序读取二进制文件的内容。这甚至可用于为其他硬件体系结构生成的 ELF 文件：

```
> readelf --file-header /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
```

```
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x402540
  Start of program headers:          64 (bytes into file)
  Start of section headers:          95720 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         32
  Section header string table index: 31
```

2.8.4 文件属性：stat

命令 **stat** 显示文件属性：

```
> stat /etc/profile
  File: `/etc/profile'
  Size: 9662            Blocks: 24         IO Block: 4096   regular file
Device: 802h/2050d      Inode: 132349      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-03-20 07:51:17.000000000 +0100
Modify: 2009-01-08 19:21:14.000000000 +0100
Change: 2009-03-18 12:55:31.000000000 +0100
```

参数 `--file-system` 生成指定文件所在的文件系统的属性细节：

```
> stat /etc/profile --file-system
  File: "/etc/profile"
    ID: d4fb76e70b4d1746 Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 2581445    Free: 1717327    Available: 1586197
Inodes: Total: 655776     Free: 490312
```

2.9 用户信息

2.9.1 访问文件的用户：`fuser`

它可用于确定当前哪些进程或用户正在访问特定的文件。例如，假设您想要卸载已挂载到 `/mnt` 的文件系统。**`umount`** 将返回“设备正忙”。然后可以使用 **`fuser`** 命令确定哪些进程正在访问该设备：

```
> fuser -v /mnt/*

                     USER        PID ACCESS COMMAND
/mnt/notes.txt       tux    26597 f....  less
```

在终止 **`less`** 进程之后（该进程在另一个终端上运行），即可成功卸载该文件系统。与 `-k` 选项结合使用时，**`fuser`** 还会停止正在访问文件的进程。

2.9.2 哪些用户在执行哪些操作：`w`

使用命令 **`w`** 可以确定谁已登录到系统，以及每个用户正在执行的操作。例如：

```
> w
 16:00:59 up 1 day,  2:41,  3 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
tux      :0       console          Wed13   ?xdm?   8:15   0.03s /usr/lib/gdm/gd
tux      console  :0               Wed13   26:41m  0.00s  0.03s /usr/lib/gdm/gd
tux      pts/0    :0               Wed13   20:11   0.10s  2.89s /usr/lib/gnome-
```

如果其他系统的任何用户已远程登录，则参数 `-f` 将显示这些用户发起连接的计算机。

2.10 时间和日期

2.10.1 使用 **time** 计量时间

使用 **time** 实用程序的命令来确定花费的时间。此实用程序有两个版本：内置 Bash 和程序 (**/usr/bin/time**)。

```
> time find . > /dev/null

real    0m4.051s ❶
user    0m0.042s ❷
sys     0m0.205s ❸
```

❶ 从命令启动到完成实际经过的时间。

❷ 由 `times` 系统调用报告的用户 CPU 时间。

❸ 由 `times` 系统调用报告的系统 CPU 时间。

`/usr/bin/time` 的输出要详细得多。建议结合使用 `-v` 开关运行此命令，以生成直观易懂的输出。

```
/usr/bin/time -v find . > /dev/null
        Command being timed: "find ."
        User time (seconds): 0.24
        System time (seconds): 2.08
        Percent of CPU this job got: 25%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.03
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 2516
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 1564
        Voluntary context switches: 36660
        Involuntary context switches: 496
```

```
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

2.11 绘制数据图表：RRDtool

您的生活中存在大量可轻松即时测量的数据。例如，温度的变化，或者您计算机网络接口发送或接收的数据量。RRDtool 可帮助您存储这些数据，并以详细的可自定义图表呈现它们。

RRDtool 适用于大多数 Unix 平台和 Linux 发行套件。SUSE® Linux Enterprise Desktop 也随附有 RRDtool。请使用 YaST 或者

以 `root` 身份在命令行中输入 **`zypper`** `install` `rrdtool` 来安装此工具。

提示：绑定

我们提供了 RRDtool 的 Perl、Python、Ruby 和 PHP 绑定，以便您能以偏好的脚本语言编写自己的监控脚本。

2.11.1 RRDtool 的工作原理

RRDtool 是**循环复用数据库工具 (Round Robin Database tool)** 的英文缩写。**循环复用**是操作恒量数据的一种方法。它采用循环缓冲区的原理，所读取的数据行既没有结尾，也没有开头。RRDtool 使用循环复用数据库来存储和读取其数据。

如前所述，RRDtool 用于处理即时变化的数据。典型案例是传感器在恒定的时间段反复读取测量的数据（例如温度、速度等），然后以指定格式导出数据。此类数据十分适合通过 RRDtool 处理，它可以轻松处理并创建所需的输出。

有时无法定期自动获取数据。需要预处理这些数据的格式才能将其提供给 RRDtool，您甚至常常需要手动操作 RRDtool。

下面是 RRDtool 基本用法的简单示例。其中演示了常规 RRDtool 工作流程的所有三个重要阶段：**创建**数据库、**更新**测量值和**查看**输出。

2.11.2 实际应用示例

假设我们要收集并查看 Linux 系统中内存用量即时变化的信息。为使示例更生动，我们将以 4 秒为间隔，测量 40 秒时间段内的当前可用内存。启动再关闭三个通常会消耗大量系统内存的应用程序：Firefox 网页浏览器、Evolution 电子邮件客户端和 Eclipse 开发框架。

2.11.2.1 收集数据

RRDtool 经常用于测量网络流量并以直观方式呈现。在这种情况下，会使用简单网络管理协议 (SNMP)。此协议可以查询网络设备的内部计数器的相关值。确切来说，这些值将通过 RRDtool 来存储。有关 SNMP 的详细信息，请参见 http://www.net-snmp.org/。

我们的情况有所不同 — 我们需要手动获取数据。一个助手脚本 **free_mem.sh** 会反复读取可用内存的当前状态，并将其写入标准输出。

```
> cat free_mem.sh
INTERVAL=4
for steps in {1..10}
do
    DATE=`date +%s`
    FREEMEM=`free -b | grep "Mem" | awk '{ print $4 }'`
    sleep $INTERVAL
    echo "rrdtool update free_mem.rrd $DATE:$FREEMEM"
done
```

- 时间间隔设置为 4 秒，这通过 **sleep** 命令来实施。
- RRDtool 接受采用特殊格式的时间信息，即所谓的 **Unix 时间**。此时间定义为自 1970 年 1 月 1 日午夜 (UTC) 开始经过的秒数。例如，1272907114 表示 2010-05-03 17:18:34。

- 可用内存信息是使用 **free** `-b` 以字节为单位报告的。将优先提供基本单位（字节）而不是倍数单位（例如 KB）。
- 带有 **echo ...** 命令的行包含数据库文件 (`free_mem.rrd`) 的未来名称，并共同创建了用于更新 RRDtool 值的命令行。

运行 **free_mem.sh** 后，您将看到如下所示的输出：

```
> sh free_mem.sh
rrdtool update free_mem.rrd 1272974835:1182994432
rrdtool update free_mem.rrd 1272974839:1162817536
rrdtool update free_mem.rrd 1272974843:1096269824
rrdtool update free_mem.rrd 1272974847:1034219520
rrdtool update free_mem.rrd 1272974851:909438976
rrdtool update free_mem.rrd 1272974855:832454656
rrdtool update free_mem.rrd 1272974859:829120512
rrdtool update free_mem.rrd 1272974863:1180377088
rrdtool update free_mem.rrd 1272974867:1179369472
rrdtool update free_mem.rrd 1272974871:1181806592
```

可以方便地使用

sh free_mem.sh > free_mem_updates.log

将命令输出重定向到某个文件以简化此命令的未来执行。

2.11.2.2 创建数据库

使用以下命令为本示例创建初始的循环复用数据库：

```
> rrdtool create free_mem.rrd --start 1272974834 --step=4 \
DS:memory:GAUGE:600:U:U RRA:AVERAGE:0.5:1:24
```

要点

- 此命令会创建名为 `free_mem.rrd` 的文件，用于在循环复用数据库中存储测量值。
- `--start` 选项指定将第一个值添加到数据库的时间（采用 Unix 时间）。在本示例中，此值比 **free_mem.sh** 输出的第一个时间值 (1272974835) 小 1。
- `--step` 指定向数据库提供测量数据的时间间隔（以秒为单位）。

- `DS:memory:GAUGE:600:U:U` 部分用于引入数据库的新数据源。此数据源名为 **memory**，类型为 **gauge**，两次更新之间的最大间隔为 600 秒，测量范围的**最小**和**最大**值未知 (U)。
- `RRA:AVERAGE:0.5:1:24` 会创建循环复用归档 (RRA)，其中存储的数据是通过用于计算数据点**平均值**的**合并函数** (CF) 处理的。合并函数的 3 个参数已追加到行尾。

如果未显示错误消息，则当前目录中已创建了 `free_mem.rrd` 数据库：

```
> ls -l free_mem.rrd
-rw-r--r-- 1 tux users 776 May  5 12:50 free_mem.rrd
```

2.11.2.3 更新数据库值

创建数据库后，需在其中填充测量数据。在第 2.11.2.1 节 “收集数据”中，我们已准备好一个包含 **`rrdtool update`** 命令的 `free_mem_updates.log` 文件。这些命令将为我们执行数据库值的更新。

```
> sh free_mem_updates.log; ls -l free_mem.rrd
-rw-r--r--  1 tux users  776 May  5 13:29 free_mem.rrd
```

可以看到，即使是在更新数据后，`free_mem.rrd` 的大小仍保持不变。

2.11.2.4 查看测量值

我们已测量了值，创建了数据库，并在其中存储了测量值。现在我们可以操作该数据库，并检索或查看其值。

要检索数据库中的所有值，请在命令行上输入以下命令：

```
> rrdtool fetch free_mem.rrd AVERAGE --start 1272974830 \
--end 1272974871
           memory
1272974832: nan
1272974836: 1.1729059840e+09
1272974840: 1.1461806080e+09
1272974844: 1.0807572480e+09
1272974848: 1.0030243840e+09
```

```
1272974852: 8.9019289600e+08
1272974856: 8.3162112000e+08
1272974860: 9.1693465600e+08
1272974864: 1.1801251840e+09
1272974868: 1.1799787520e+09
1272974872: nan
```

要点

- AVERAGE 会从数据库提取平均值点，因为只通过 AVERAGE 处理定义了一个数据源（第 2.11.2.2 节 “创建数据库”），没有其他函数可用。
- 输出的第一行列显第 2.11.2.2 节 “创建数据库”中定义的数据源的名称。
- 左侧结果列表示各个时间点，右侧结果列以科学记数法表示对应的测量平均值。
- 最后一行中的 nan 表示“不是数字”。

现在将绘制一个图表来表示数据库中存储的值：

```
> rrdtool graph free_mem.png \
--start 1272974830 \
--end 1272974871 \
--step=4 \
DEF:free_memory=free_mem.rrd:memory:AVERAGE \
LINE2:free_memory#FF0000 \
--vertical-label "GB" \
--title "Free System Memory in Time" \
--zoom 1.5 \
--x-grid SECOND:1:SECOND:4:SECOND:10:0:%X
```

要点

- free_mem.png 是要创建的图表的文件名。
- --start 和 --end 限制要绘制的图表的时间范围。
- --step 指定图表的时间解析度（以秒为单位）。
- DEF:... 部分是名为 **free_memory** 的数据定义。其数据会从 free_mem.rrd 数据库读取，其数据源名为 **memory**。系统将计算**平均**值点，因为第 2.11.2.2 节 “创建数据库”中未定义其他值点。

- `LINE...` 部分指定要绘制到图表中的线条的属性。其宽度为 2 像素，数据来自 **free_memory** 定义，颜色为红色。
- `--vertical-label` 设置要在 **y** 轴上列显的标签，`--title` 设置整个图表的主标签。
- `--zoom` 指定图表的缩放系数。此值必须大于零。
- `--x-grid` 指定如何在图表中绘制网格线及其标签。本示例每隔一秒绘制一次网格线，每隔 4 秒绘制一次主网格线。每隔 10 秒将标签绘制在主网格线下。

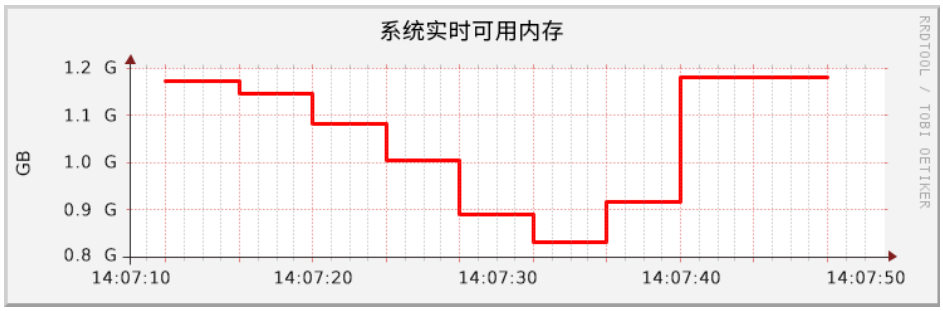

图 2.1： 使用 RRDTOOL 创建的示例图表

2.11.3 更多信息

RRDtool 是一个复杂的工具，它附带了许多子命令和命令行选项。其中一些命令和选项很容易理解，但要使其生成所需的结果并根据您的喜好对其进行微调，可能还要花费许多功夫。

除了提供基本信息的 RRDtool 的手册页 (**`man 1 rrdtool`**)，您还应该查看 RRDtool 主页 (https://oss.oetiker.ch/rrdtool/)。**`rrdtool`** 命令及其所有子命令有详细的文档 (https://oss.oetiker.ch/rrdtool/doc/index.en.html) 可供参考。此外，还有几篇教程 (https://oss.oetiker.ch/rrdtool/tut/index.en.html) 可帮助您理解常用的 RRDtool 工作流程。

如果您想要监控网络流量，请查看 MRTG (Multi Router Traffic Grapher) (https://oss.oetiker.ch/mrtg/)。MRTG 可以绘制许多网络设备的活动图表。它可以使用 RRDtool。

3 系统日志文件

系统日志文件分析是分析系统时最重要的任务之一。事实上，在对系统进行维护或查错时，第一件事就是要查看系统日志文件。SUSE Linux Enterprise Desktop 会自动详细记录系统上发生的几乎一切事件。自从过渡到 systemd 之后，内核消息以及已在 systemd 中注册的系统服务的消息都将记录在 systemd 日记中（请参见《管理指南》, 第 21 章 “**journalctl**：查询 systemd 日记”）。其他日志文件（主要是系统应用程序的日志文件）将以纯文本格式写入，并可使用编辑器或分页器轻松阅读。还可以使用脚本来分析这些文件。这可让您过滤其内容。

3.1 /var/log/ 中的系统日志文件

系统日志文件始终位于 /var/log 目录下。以下列表提供了在完成 SUSE Linux Enterprise Desktop 的默认安装后其上所有系统日志文件的概述。根据您的安装范围，/var/log 还会包含来自此处未列出的其他服务和应用程序的日志文件。下面所述的某些文件和目录以“占位符”表示，仅当安装了相应的应用程序时才会使用它们。大多数日志文件仅对 root 用户可见。

apparmor/
: AppArmor 日志文件。有关 AppArmor 的详细信息，请参见《安全和强化指南》。

audit/
: 来自审计框架的日志。有关详细信息，请参见《安全和强化指南》。

ConsoleKit/
: ConsoleKit 守护程序的日志。该守护程序用于跟踪哪些用户已登录，以及他们如何与计算机交互。

cups/
: 通用 Unix 打印系统 (cups) 的访问和错误日志。

firewall
: 防火墙日志。

gdm/
: 来自 GNOME 显示管理器的日志文件。

`krb5/`

来自 Kerberos 网络身份验证系统的日志文件。

`lastlog`

包含每个用户上次登录相关信息的数据库。可使用命令 **`lastlog`** 进行查看。有关更多信息，请参见**`man 8 lastlog`**。

`localmessages`

某些引导脚本的日志消息，例如，DHCP 客户端的日志。

`mail*`

邮件服务器（`sendmail`、`postfix`）日志。

`messages`

这是所有内核和系统日志消息的默认保存位置，出现问题时应首先查看该位置（以及 `/var/log/warn`）。

`NetworkManager`

NetworkManager 日志文件。

`news/`

来自新闻服务器的日志消息。

`chrony/`

来自网络时间协议守护程序 (`chrony`) 的日志。

`pk_backend_zypp*`

PackageKit（使用 `libzypp` 后端）日志文件。

`samba/`

来自 Samba、Windows SMB/CIFS 文件服务器的日志文件。

`warn`

所有系统警告和错误的日志。出现问题时，应首先查看此位置（以及 `systemd` 日记的输出）。

`wtmp`

所有登录/注销活动和远程连接的数据库。可使用命令 **`last`** 进行查看。有关更多信息，请参见**`man 1 last`**。

`Xorg.NUMBER.log`

X.Org 启动日志文件。如果在启动 X.Org 时遇到问题，请参见这些文件。文件名中的 `NUMBER` 是显示编号。例如，默认的 `Xorg.0.log` 是显示编号 `0` 的日志，`Xorg.1.log` 是显示编号 `1` 的日志。先前的 X.Org 启动事件产生的日志文件按照 `Xorg.NUMBER.log.old` 命名。

注意

仅当您以 `root` 身份启动 X.Org 会话时，`/var/log/` 目录中才会生成 X.Org 日志文件。如果以任何其他用户的身份启动 X.Org 会话，则可以在 `~/.local/share/xorg/` 目录中找到日志文件。

`YaST2/`

所有 YaST 日志文件。

`zypp/`

`libzypp` 日志文件。可参考这些文件获取软件包安装历史。

`zypper.log`

来自命令行安装程序 **`zypper`** 的日志。

3.2 查看和分析日志文件

要查看日志文件，可以使用任何文本编辑器。此外，还可以使用一个简单的 YaST 模块来查看 YaST 控制中心的杂项 › 系统日志下提供的系统日志。

要在文本控制台中查看日志文件，请使用命令 **`less`** 或 **`more`**。使用 **`head`** 和 **`tail`** 可查看日志文件的开头和结尾。要实时查看追加到日志文件的条目，请使用 **`tail`** `-f`。有关如何使用这些工具的信息，请查看它们的手册页。

要在日志文件中搜索字符串或正则表达式，请使用 **`grep`**。**`awk`** 可以用于解析和重写日志文件。

3.3 使用 logrotate 管理日志文件

/var/log 下的日志文件每天都会增长，很快就会变得庞大。**logrotate** 是一种可帮助您管理日志文件并控制其增长速度的工具。使用此工具可以自动轮换、去除、压缩日志文件，以及通过邮件发送这些文件。可以定期（每日、每周或每月）或者在超过特定大小时处理日志文件。

logrotate 每天由 systemd 运行，因此前者每天只会修改一次日志文件。但存在一些例外情况：因超过大小限制而修改日志文件时、如果一天内多次运行 **logrotate**，或者如果启用 --force。使用 /var/lib/misc/logrotate.status 可确定上次轮换特定文件的时间。

logrotate 的主配置文件为 /etc/logrotate.conf。生成日志文件的系统软件包和程序（例如 apache2）将其自身的配置文件放在 /etc/logrotate.d/ 目录中。/etc/logrotate.d/ 的内容是通过 /etc/logrotate.conf 包含的。

例 3.1：/etc/logrotate.conf 的示例

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# comment these to switch compression to use gzip or another
# compression scheme
compresscmd /usr/bin/bzip2
uncompresscmd /usr/bin/bunzip2

# RPM packages drop log rotation information into this directory
```

```
include /etc/logrotate.d
```

重要：避免权限冲突

`create` 选项会关注 `/etc/permissions*` 中所指定文件的模式和所有权。如果您修改这些设置，请确保不会引发冲突。

3.4 使用 **logwatch** 监控日志文件

logwatch 是一个可自定义且可插入的日志监控脚本。它可以分析系统日志、提取重要信息，并以直观易懂的方式显示这些信息。要使用 **logwatch**，请安装 `logwatch` 软件包。

可在命令行上使用 **logwatch** 来生成动态报告，或通过 `cron` 来定期创建自定义报告。报告可在屏幕上列显、保存到文件，或通过邮件发送到指定的地址。通过 `cron` 自动生成报告时，后一种做法特别有用。

在命令行上，可以告知 **logwatch** 要针对哪个服务和时间范围生成报告，以及所包含信息的详细程度：

```
# Detailed report on all kernel messages from yesterday
logwatch --service kernel --detail High --range Yesterday --print

# Low detail report on all sshd events recorded (incl. archived logs)
logwatch --service sshd --detail Low --range All --archives --print

# Mail a report on all smartd messages from May 5th to May 7th to root@localhost
logwatch --service smartd --range 'between 5/5/2005 and 5/7/2005' \
--mailto root@localhost --print
```

`--range` 选项的语法比较复杂 — 有关细节，请参见 **logwatch** `--range help`。使用以下命令获取可查询的所有服务的列表：

```
> ls /usr/share/logwatch/default.conf/services/ | sed 's/\.conf//g'
```

logwatch 的可自定义程度很高。不过，默认配置应该足以满足需求。默认配置文件位于 `/usr/share/logwatch/default.conf/` 下。切勿更改这些文件，因为下一次更新时会再次重写它们。应将自定义配置放入 `/etc/logwatch/conf/`（但您可以使用默认配置文件作为模板）。有关如何自定义 **logwatch** 的详细操作指南，请参见 `/usr/share/doc/packages/logwatch/HOWTO-Customize-LogWatch`。存在以下配置文件：

`logwatch.conf`
: 主要配置文件。默认版本带有大量注释。可在命令行重写每个配置选项。

`ignore.conf`
: 过滤掉 **`logwatch`** 应全局忽略的所有行。

`services/*.conf`
: 该服务目录包含了可为其生成报告的每个服务的配置文件。

`logfiles/*.conf`
: 有关应分析每个服务的哪些日志文件的规范。

3.5 为 root 配置邮件转发

系统守护程序、`cron` 作业、`systemd` 计时器和其他应用程序可以生成邮件并将其发送给系统的 `root` 用户。默认情况下，每个用户帐户拥有一个本地邮箱，在登录后会收到有关新邮件的通知。

这些邮件可能包含需要系统管理员迅速应对的安全相关报告和事件。要及时收到有关这些邮件的通知，强烈建议将这些邮件转发到定期检查的专用远程电子邮件帐户。

过程 3.1： 为 root 用户配置邮件转发

要为 `root` 用户转发邮件，请执行以下步骤：

1. 安装 `yast2-mail` 软件包：

```
# zypper in yast2-mail
```

2. 运行交互式 YaST 邮件配置：

```
# yast mail
```

3. 对于连接类型，选择永久，然后单击下一步继续。
4. 输入外发邮件服务器的地址。根据需要配置身份验证。强烈建议强制实施 TLS 加密，以防通过网络发送未经加密的潜在敏感系统数据。单击下一步。
5. 输入要将 root 用户的邮件转发到的电子邮件地址，然后完成配置。

> **重要：不要接受远程 SMTP 连接**
>
> **不要**启用接受远程 SMTP 连接，否则本地计算机将充当邮件中继。

6. 发送一封邮件以测试邮件是否正常转发：

```
> mail root
subject: test
test
.
```

7. 使用 **mailq** 命令校验是否已发送测试邮件。如果成功，则队列应该为空。该邮件应该已由事先配置的专用邮件地址接收。

根据受管的计算机数和需要收到系统事件通知的人数，可以建立不同的电子邮件地址模型：

- 在只能由一个人访问的电子邮件帐户中收集来自不同系统的邮件。
- 在可由所有相关人员访问的组电子邮件帐户（别名或邮件列表）中收集来自不同系统的邮件。
- 为每个系统创建独立的电子邮件帐户。

管理员必须定期检查相关的电子邮件帐户，这一点至关重要。为了简化此项工作并识别重要事件，请避免发送不必要的信息。将应用程序配置为仅发送相关信息。

3.6 将日志消息转发到中心系统日志服务器

可将系统日志数据从各个系统转发到网络上的中心系统日志服务器。这样，管理员便可以大致了解所有主机上的事件，并防止成功攻克了系统的攻击者操控系统日志以掩盖其踪迹。

设置中心系统日志服务器的过程由两个部分组成。首先配置中心日志服务器，然后配置客户端以进行远程日志记录。

3.6.1 设置中心系统日志服务器

过程 3.2： 配置中心 rsyslog 服务器

要设置中心系统日志服务器，请执行以下步骤：

1. 编辑配置文件 `/etc/rsyslog.d/remote.conf`。
2. 取消注释配置文件的 `UDP Syslog Server` 或 `TCP Syslog Server` 部分中的以下行。为 `rsyslogd` 分配 IP 地址和端口。

   TCP 示例：

   ```
   $ModLoad imtcp.so
   $UDPServerAddress IP ❶
   $InputTCPServerRun PORT ❷
   ```

   UDP 示例：

   ```
   $ModLoad imudp.so
   $UDPServerAddress IP ❶
   $UDPServerRun PORT ❷
   ```

   ❶ 要侦听的 `rsyslogd` 接口的 IP 地址。如果不指定地址，守护程序将侦听所有接口。

   ❷ 要侦听的 `rsyslogd` 端口。选择低于 1024 的特权端口。默认值为 514。

   重要：TCP 与 UDP 协议

   在传统上，系统日志会使用 UDP 协议通过网络来传输日志消息。这种传输方式的开销较少，但可靠性不足。在负载较高的情况下，日志消息可能会丢失。

   TCP 协议更可靠，因此应该优先使用它而不是 UDP。

   注意：将 UDPServerAddress 用于 TCP

   TCP 示例中的 `$UDPServerAddress` 配置参数并没有错误。尽管其名称包含 UDP，但它同时用于 TCP 和 UDP。

3. 保存文件。

4. 重启动 `rsyslog` 服务：

```
> sudo systemctl restart rsyslog.service
```

5. 在防火墙中打开相应的端口。对于与 TCP 配合使用的 `firewalld`，请在端口 514 上运行：

```
> sudo firewall-cmd --add-port 514/tcp --permanent
> sudo firewall-cmd --reload
```

现在，您已配置中心系统日志服务器。接下来，请配置客户端以进行远程日志记录。

3.6.2 设置客户端计算机

过程 3.3： 配置一个用于进行远程日志记录的 RSYSLOG 实例

要配置计算机以将日志远程记录到中心系统日志服务器，请执行以下步骤：

1. 编辑配置文件 `/etc/rsyslog.d/remote.conf`。
2. 取消注释相应的行（TCP 或 UDP），并将 `remote-host` 替换为在第 3.6.1 节 “设置中心系统日志服务器”中设置的中心系统日志服务器地址。

   TCP 示例：

```
# Remote Logging using TCP for reliable delivery
# remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
*.* @@remote-host
```

   UDP 示例：

```
# Remote Logging using UDP
# remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
*.* @remote-host
```

3. 保存文件。
4. 重启动 `rsyslog` 服务：

```
> sudo systemctl restart rsyslog.service
```

5. 校验系统日志转发是否可正常进行：

```
> logger "hello world"
```

中心系统日志服务器上现在应会出现日志消息 `hello world`。

现在，您已配置一个系统以将日志远程记录到中心系统日志服务器。针对应该远程记录其日志的所有系统重复上述过程。

3.6.3 更多信息

此基本设置不包括加密，且仅适用于可信的内部网络。强烈建议使用 TLS 加密，但这需要一个证书基础架构。

在此配置中，来自远程主机的所有消息的处理方式与在中心系统日志服务器上相同。请考虑按远程主机将消息过滤到单独的文件中，或者按消息类别将消息分类。

有关加密、过滤和其他高级主题的详细信息，请参见 https://www.rsyslog.com/doc/master/index.html#manual 上的 RSyslog 文档。

3.7 使用 `logger` 创建系统日志项

`logger` 是用于在系统日志中创建日志项的工具。它提供了 rsyslogd 系统日志模块的外壳命令接口。例如，下面一行命令会在 `/var/log/messages` 中或直接在日记（如果未运行日志记录工具）输出其消息：

```
> logger -t Test "This message comes from $USER"
```

根据当前用户和主机名，日志将包含如下一行：

```
Sep 28 13:09:31 venus Test: This message comes from tux
```

III 内核监控

4 SystemTap — 过滤和分析系统数据 71

5 内核探测 85

6 使用 Perf 进行基于硬件的性能监控 90

7 OProfile — 系统范围的分析器 95

8 动态调试 - 内核调试消息 101

4 SystemTap — 过滤和分析系统数据

SystemTap 提供了命令行界面和脚本语言，用于细致检查运行中 Linux 系统（特别是内核）的活动。SystemTap 脚本采用 SystemTap 脚本语言编写，随后会编译为 C 代码内核模块并插入到内核中。您可以设计脚本来提取、过滤和汇总数据，以便对复杂的性能问题或功能问题进行诊断。SystemTap 提供的信息与 **`netstat`**、**`ps`**、**`top`** 和 **`iostat`** 等工具的输出类似。不过，它提供了更多用于过滤和分析所收集信息的选项。

4.1 概念概述

每当您运行 SystemTap 脚本时，都会启动一个 SystemTap 会话。需要针对脚本传递几个参数，脚本才能运行。然后，该脚本会编译为内核模块并加载到内核中。如果以前执行过该脚本，并且任何系统组件都未更改（例如，不同的编译器或内核版本、库路径或脚本内容），SystemTap 不会再次编译该脚本，而是使用 SystemTap 缓存 (`~/.systemtap`) 中存储的 `*.c` 和 `*.ko` 数据。

当 tap 完成运行时，将卸载模块。有关示例，请参见第 4.2 节 “安装和设置”中的测试运行和相关说明。

4.1.1 SystemTap 脚本

SystemTap 的用法基于 SystemTap 脚本 (`*.stp`)。这些脚本会告知 SystemTap 要收集哪类信息，以及收集信息后要执行哪项操作。脚本采用与 AWK 和 C 类似的 SystemTap 脚本语言编写。有关语言定义，请访问 https://sourceware.org/systemtap/langref.pdf。您可以在 https://www.sourceware.org/systemtap/examples/ 中找到大量实用的示例脚本。

SystemTap 脚本的基本运作原理是指定 `events` 并为其提供 `handlers`。当 SystemTap 运行脚本时，它会监控特定的事件。发生某个事件时，Linux 内核会将处理程序作为子例程运行，然后继续。因此，事件充当了运行处理程序的触发器。处理程序可以记录指定的数据，并以特定的方式列显数据。

SystemTap 语言仅使用少量几种数据（整数、字符串，以及这些类型的关联数组）和完整控制结构（块、条件、循环、函数）。它包含轻量标点符号（可选用分号），且不需要详细的声明（系统会自动推断并检查类型）。

有关 SystemTap 脚本及其语法的详细信息，请参见第 4.3 节 “脚本语法”，以及 `systemtap-docs` 软件包中提供的 **`stapprobes`** 和 **`stapfuncs`** 手册页。

4.1.2 Tapset

是预先编写、可在 SystemTap 脚本中使用的探测和函数库。当用户运行某个 SystemTap 脚本时，SystemTap 会根据 tapset 库检查该脚本的探测事件和处理程序。然后，SystemTap 会先加载相应的探测和函数，然后再将脚本转换为 C。与 SystemTap 脚本本身类似，tapset 使用文件扩展名 `*.stp`。

但是，与 SystemTap 脚本不同的是，tapset 不可直接执行。它们构成了可供其他脚本提取定义的库。因此，tapset 库是一个抽象层，旨在方便用户定义事件和函数。Tapset 为用户可能想要作为事件指定的函数提供别名。了解正确的别名通常比记住特定的内核函数更容易（不同内核版本的内核函数可能会不同）。

4.1.3 命令和特权

与 SystemTap 关联的主要命令包括 **`stap`** 和 **`staprun`**。要执行这些命令，您需要拥有 `root` 特权，或者必须是 `stapdev` 或 `stapusr` 组的成员。

`stap`

SystemTap 前端。运行 SystemTap 脚本（通过文件或标准输入）。此命令会将该脚本转换为 C 代码，并将生成的内核模块加载到正在运行的 Linux 内核中。然后执行请求的系统跟踪或探测函数。

`staprun`

SystemTap 后端。加载和卸载 SystemTap 前端生成的内核模块。

如需每个命令的选项列表，请使用 `--help`。有关细节，请参见 **`stap`** 和 **`staprun`** 手册页。

为了避免单纯出于让用户能够使用 SystemTap 的目的而向其授予 `root` 访问权限，请使用以下 SystemTap 组之一。SUSE Linux Enterprise Desktop 上默认未提供这些组，但您可以创建组，并相应地修改访问权限。另外，您还可调整 **`staprun`** 命令的权限，前提是这不会对您的环境安全产生不当影响。

`stapdev`

此组的成员可以使用 **`stap`** 运行 SystemTap 脚本，或使用 **`staprun`** 运行 SystemTap 工具模块。由于运行中的 **`stap`** 涉及到将脚本编译为内核模块并将其加载到内核中，此组的成员仍拥有有效的 `root` 访问权限。

`stapusr`

此组的成员只能使用 **`staprun`** 运行 SystemTap 工具模块。此外，他们只能通过 `/lib/modules/KERNEL_VERSION/systemtap/` 运行这些模块。此目录必须由 `root` 拥有，且仅供 `root` 用户写入。

4.1.4 重要文件和目录

以下列表概述了 SystemTap 主要文件和目录。

`/lib/modules/KERNEL_VERSION/systemtap/`

保存 SystemTap 工具模块。

`/usr/share/systemtap/tapset/`

保存标准的 tapset 库。

`/usr/share/doc/packages/systemtap/examples`

保存用于不同目的的多个示例 SystemTap 脚本。仅当已安装 `systemtap-docs` 软件包时才可用。

`~/.systemtap/cache`

缓存的 SystemTap 文件的数据目录。

`/tmp/stap*`

SystemTap 文件的临时目录，包含已转换的 C 代码和内核对象。

4.2 安装和设置

由于 SystemTap 需要内核相关信息，必须安装一些额外的内核相关软件包。对于您要使用 SystemTap 探测的每个内核，需要安装下面一组软件包。这组软件包应该与内核版本和变种（在下面的概述中以 `*` 表明）完全匹配。

重要：包含调试信息的软件包的储存库

如果您为系统订阅了联机更新，便可以在 SUSE Linux Enterprise Desktop 15 SP7 的相关 `*-Debuginfo-Updates` 联机安装储存库中找到 “debuginfo” 软件包。使用 YaST 启用该储存库。

对于经典 SystemTap 设置，请安装以下软件包（使用 YaST 或 **`zypper`**）。

- `systemtap`
- `systemtap-server`
- `systemtap-docs` (可选)
- `kernel-*-base`
- `kernel-*-debuginfo`
- `kernel-*-devel`
- `kernel-source-*`
- `gcc`

要访问手册页和用于不同目的的有用示例 SystemTap 脚本集合，另外还需安装 `systemtap-docs` 软件包。

要检查是否在计算机上正确安装了所有软件包以及是否已可使用 SystemTap，请以 `root` 身份执行以下命令。

```
# stap -v -e 'probe vfs.read {printf("read performed\n"); exit()}'
```

此命令通过运行一个脚本并返回输出，来探测当前使用的内核。如果输出类似于以下内容，则表示 SystemTap 已成功部署并可供使用：

```
Pass ❶: parsed user script and 59 library script(s) in 80usr/0sys/214real ms.
Pass ❷: analyzed script: 1 probe(s), 11 function(s), 2 embed(s), 1 global(s) in
 140usr/20sys/412real ms.
Pass ❸: translated to C into
 "/tmp/stapDwEk76/stap_1856e21ea1c246da85ad8c66b4338349_4970.c" in 160usr/0sys/408real ms.
Pass ❹: compiled C into "stap_1856e21ea1c246da85ad8c66b4338349_4970.ko" in
```

```
 2030usr/360sys/10182real ms.
Pass ❺: starting run.
 read performed
Pass ❺: run completed in 10usr/20sys/257real ms.
```

❶ 根据任何所用 tapset 的 `/usr/share/systemtap/tapset/` 中的现有 tapset 库检查脚本。Tapset 是由脚本组成的库，库中包含可在 SystemTap 脚本中使用的预先编写的探测和函数。

❷ 检查脚本的各个组成部分。

❸ 将脚本转换为 C。运行系统 C 编译器以基于脚本创建内核模块。生成的 C 代码 (`*.c`) 和内核模块 (`*.ko`) 都存储在 SystemTap 缓存 `~/.systemtap` 中。

❹ 在脚本中通过挂接到内核来加载模块并启用所有探测（事件和处理程序）。探测的事件是虚拟文件系统 (VFS) 读取操作。当该事件在任何处理器上发生时，一个有效的处理程序会执行（列显文本 `read performed`），随后会正常关闭（未出现任何错误）。

❺ 终止 SystemTap 会话后，已禁用探测并卸载内核模块。

如果在测试期间出现了任何错误消息，请检查输出以获取有关缺少哪些软件包的提示，并确保正确安装这些软件包。另外，可能还需要重引导系统并加载适当内核。

4.3 脚本语法

SystemTap 脚本包含以下两个组成部分：

SystemTap 事件（探测点）

为要执行的关联处理程序上的内核事件命名。事件的示例包括进入或退出特定的函数、计时器即将失效，或者启动或终止会话。

SystemTap 处理程序（探测主体）

用于指定每当发生特定事件时要执行操作的脚本语言语句系列。通常包括从事件环境提取数据、将数据存储到内部变量，或列显结果。

事件及其相应的处理程序统称为 `probe`。SystemTap 事件也称为`probe points`。探测的处理程序也称为`probe body`。

您可以在 SystemTap 脚本中的任意位置插入不同样式的注释：使用 `#`、`/* */` 或 `//` 作为标记。

4.3.1 探测格式

一个 SystemTap 脚本可以包含多个探测。必须采用以下格式编写探测：

```
probe EVENT {STATEMENTS}
```

每个探测有一个对应的语句块。此语句块必须括在 { } 中，并包含要针对每个事件执行的语句。

例 4.1： 简单 SYSTEMTAP 脚本

以下示例演示了一个简单的 SystemTap 脚本。

```
probe❶ begin❷
{❸
   printf❹ ("hello world\n")❺
   exit ()❻
}❼
```

❶ 探测开始。

❷ 事件 `begin`（SystemTap 会话开始）。

❸ 处理程序定义开始，以 { 指示。

❹ 处理程序中定义的第一个函数：`printf` 函数。

❺ `printf` 函数列显的字符串，后接换行符 `/n`。

❻ 处理程序中定义的第二个函数：`exit()` 函数。请注意，SystemTap 脚本将一直运行到执行 `exit()` 函数为止。如果您想让脚本提前停止执行，请按 `Ctrl` – `C` 手动停止。

❼ 处理程序定义结束，以 } 指示。

事件 `begin` ❷（SystemTap 会话开始）会触发 { } 中封装的处理程序。在本例中，该处理程序是 `printf` 函数 ❹。在本例中，该函数列显 `hello world` 后接换行符 ❺。然后脚本退出。

如果您的语句块包含多个语句，SystemTap 将按顺序执行这些语句 — 您无需在多个语句之间插入特殊分隔符或终止符。还可以将一个语句块嵌套在另一个语句块中。一般情况下，SystemTap 脚本中的语句块使用的语法和语义与 C 编程语言中使用的相同。

4.3.2 SystemTap 事件（探测点）

SystemTap 支持多个内置事件。

一般事件语法是带点符号序列。这可将事件名称空间分成不同的部分。可以使用类似于函数调用的语法，通过字符串或数字文本将每个组成部分标识符参数化。组成部分可以包含 * 字符以扩展到其他匹配的探测点。探测点后可跟 ? 字符，表示该探测点为可选项，且即使无法扩展也不应报错。此外，探测点后可跟 ! 字符，表示该探测点既是可选项，又是充分条件。

SystemTap 支持每个探测有多个事件 — 需以逗号 (,) 分隔这些事件。如果在一个探测中指定了多个事件，当发生任何指定事件时，SystemTap 将执行处理程序。

事件可分为以下类别：

- 同步事件：当任一进程在内核代码中的特定位置执行指令时发生。它为其他事件提供了一个参照点（指令地址），从中可以获得更多环境数据。
  `vfs.FILE_OPERATION` 就是一个同步事件：这是虚拟文件系统 (VFS) 中 `FILE_OPERATION` 事件的入口。例如，在第 4.2 节 “安装和设置”中，`read` 就是 VFS 所使用的 `FILE_OPERATION` 事件。
- 异步事件：不与代码中的特定指令或位置相关联。此探测点系列主要包含计数器、计时器和类似构造。
  异步事件的示例包括：`begin`（SystemTap 会话开始） — 运行 SystemTap 脚本时；`end`（SystemTap 会话结束）或计时器事件。计时器事件指定要定期执行的处理程序，例如 `example timer.s(SECONDS)` 或 `timer.ms(MILLISECONDS)`。
  与其他收集信息的探测一起使用时，计时器事件可让您列显定期更新，了解这些信息在一段时间内的变化。

例 4.2： 包含计时器事件的探测

> 例如，以下探测每隔 4 秒会列显一次文本 “hello world”：
>
> ```
> probe timer.s(4)
> {
>    printf("hello world\n")
> }
> ```

有关支持的事件的详细信息，请参见 **`stapprobes`** 手册页。该手册页的 See Also 部分还包含了其他手册页的链接，其中讨论了特定子系统和组件的支持事件。

4.3.3 SystemTap 处理程序（探测主体）

每个 SystemTap 事件附带一个对应的处理程序，该处理程序是为该事件定义的，包含一个语句块。

4.3.3.1 函数

如果您需要在多个探测中使用相同的语句集，可将这些语句放到一个函数中，以方便重复使用。函数是由关键字 `function` 后接名称定义的。函数接受任意数量的字符串或数字参数（以值的形式指定），可返回单个字符串或数字。

```
function FUNCTION_NAME(ARGUMENTS) {STATEMENTS}
probe EVENT {FUNCTION_NAME(ARGUMENTS)}
```

执行 `EVENT` 的探测时，将执行 `FUNCTION_NAME` 中的语句。`ARGUMENTS` 是传入函数的可选值。

可在脚本中的任意位置定义函数。函数可以接受任意

例 4.1 “简单 SystemTap 脚本”中已介绍了一个常用函数：`printf` 函数，用于列显带格式的数据。使用 `printf` 函数时，您可以使用格式字符串指定要列显参数的方式。格式字符串括在引号中，可以包含进一步的格式说明符（以 `%` 字符引入）。

要使用哪些格式字符串取决于您的参数列表。格式字符串可以包含多个格式说明符 — 每个说明符与相应的参数相匹配。可使用逗号分隔多个参数。

例 4.3： 带格式说明符的 `printf` 函数

```
printf ("❶%s❷(%d❸) open\n❹", execname(), pid())
```

❶ 格式字符串的开头，以 `"` 指示。

❷ 字符串格式说明符。

❸ 整数格式说明符。

❹ 格式字符串的末尾，以 `"` 指示。

上述示例以字符串形式列显当前可执行文件名 (`execname()`)，并以整数（括在方括号中）形式列显进程 ID (`pid()`)。然后，依次列显一个空格、单词 `open` 和一个换行符，如下所示：

```
[...]
vmware-guestd(2206) open
held(2360) open
[...]
```

除了例 4.3 “带格式说明符的 printf 函数”中使用的两个函数（execname() 和 pid()）以外，还可将其他各种函数用作 printf 参数。

最常用的 SystemTap 函数如下：

tid()

当前线程的 ID。

pid()

当前线程的进程 ID。

uid()

当前用户的 ID。

cpu()

当前 CPU 编号。

execname()

当前进程的名称。

gettimeofday_s()

自 Unix 纪元（1970 年 1 月 1 日）起经过的秒数。

ctime()

将时间转换为字符串。

pp()

用于描述当前正在处理的探测点的字符串。

thread_indent()

用于组织列显结果的有用函数。它会（在内部）存储每个线程 (tid()) 的缩进计数器。该函数接受一个参数，即缩进增量，用于指示要在线程的缩进计数器中添加或去除多少个空格。它会返回一个字符串，其中包含一些泛型跟踪数据，以及相应的缩进空格数。返回的泛型数据包括时戳（自线程初始缩进以来的微秒数）、进程名称和线程 ID 本身。这样您便可以识别调用了哪些函数、谁调用了这些函数，以及花费了多长时间。

调用入口和出口通常不会彼此紧靠（否则很容易匹配）。在第一个调用入口及其出口之间，通常还创建了其他调用入口和出口。缩进计数器可帮助您将一个入口与对应的出口进行匹配，当后续函数调用**并非**前一个函数的出口时，缩进计数器会对该后续函数调用进行缩进。

有关支持的 SystemTap 函数的详细信息，请参见 **`stapfuncs`** 手册页。

4.3.3.2 其他基本构造

除函数外，您还可以在 SystemTap 处理程序中使用其他常见构造，包括变量、条件语句（例如 `if`/`else`、`while` 循环、`for` 循环）、数组或命令行参数。

4.3.3.2.1 变量

可在脚本中的任意位置定义变量。要定义变量，只需选择一个名称，并通过函数或表达式为其赋值：

```
foo = gettimeofday( )
```

然后便可以在表达式中使用该变量。SystemTap 根据变量的赋值类型自动推断每个标识符的类型（字符串或数字）。任何不一致性都将报告为错误。在上述示例中，`foo` 将自动分类为数字，可通过 `printf()` 并结合使用整数格式说明符 (%d) 进行列显。

不过，默认情况下，变量位于包含它们的探测本地。系统每次调用处理程序时，会初始化、使用并处置这些变量。要在两个探测之间共享变量，请在脚本中的任意位置将其声明为全局变量。为此，请在探测的外部使用 `global` 关键字：

例 4.4：使用全局变量

```
global count_jiffies, count_ms
probe timer.jiffies(100) { count_jiffies ++ }
probe timer.ms(100) { count_ms ++ }
probe timer.ms(12345)
{
  hz=(1000*count_jiffies) / count_ms
  printf ("jiffies:ms ratio %d:%d => CONFIG_HZ=%d\n",
    count_jiffies, count_ms, hz)
  exit ()
```

```
}
```

此示例脚本使用用于统计 jiffy 和毫秒数然后进行相应计算的计时器来计算内核的 CONFIG_HZ 设置。（Jiffy 是指一次系统计时器中断的持续时间。它不是一个绝对时间间隔单位，因为其持续时间取决于特定硬件平台的时钟中断频率）。借助 `global` 语句，还可以在探测 `timer.ms(12345)` 中使用变量 `count_jiffies` 和 `count_ms`。指定 `++` 时，变量值将会加 `1`。

4.3.3.2.2 条件语句

可以在 SystemTap 脚本中使用多种条件语句。最常见的条件语句如下：

if/else 语句

这些语句使用以下格式表达：

```
if (CONDITION) ❶ STATEMENT1 ❷
else ❸ STATEMENT2 ❹
```

`if` 语句将整数值表达式与零进行比较。如果条件表达式 ❶ 不为零，则执行第一个语句 ❷。如果条件表达式为零，则执行第二个语句 ❹。else 子句（❸ 和 ❹）是可选子句。❷ 和 ❹ 也可以是语句块。

While 循环

这些语句使用以下格式表达：

```
while (CONDITION) ❶ STATEMENT ❷
```

当 `condition` 不为零时，将执行语句 ❷。❷ 也可以是一个语句块。它必须更改某个值，以便 `condition` 最终变为零。

For 循环

这些语句是 `while` 循环的快捷方式，使用以下格式表达：

```
for (INITIALIZATION ❶; CONDITIONAL ❷; INCREMENT ❸) statement
```

❶ 中指定的表达式用于初始化循环迭代次数的计数器，在开始执行循环之前执行。循环执行将持续到循环条件 ❷ 为 false。（在每个循环迭代的开头检查此表达式）。❸ 中指定的表达式用于递增循环计数器。它在每个循环迭代的末尾执行。

条件运算符

可在条件语句中使用以下运算符：

==: 等于

!=: 不等于

>=: 大于等于

<=: 小于等于

4.4 示例脚本

如果您已安装 `systemtap-docs` 软件包，可以在 `/usr/share/doc/packages/systemtap/examples` 中找到几个有用的 SystemTap 示例脚本。

本节将详细介绍一个相当简单的示例脚本：`/usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp`。

例 4.5： 使用 `tcp_connections.stp` 监控传入的 TCP 连接

```
#! /usr/bin/env stap

probe begin {
  printf("%6s %16s %6s %6s %16s\n",
         "UID", "CMD", "PID", "PORT", "IP_SOURCE")
}

probe kernel.function("tcp_accept").return?,
      kernel.function("inet_csk_accept").return? {
  sock = $return
  if (sock != 0)
    printf("%6d %16s %6d %6d %16s\n", uid(), execname(), pid(),
           inet_get_local_port(sock), inet_get_ip_source(sock))
}
```

此 SystemTap 脚本可监控传入的 TCP 连接，并帮助您实时识别未经授权或不必要的网络访问请求。它会显示计算机接受的每个新传入 TCP 连接的以下信息：

- 用户 ID (UID)
- 接受连接的命令 (CMD)
- 命令的进程 ID (PID)
- 连接使用的端口 (PORT)
- TCP 连接的来源 IP 地址 (IP_SOURCE)

要运行该脚本，请执行

```
stap /usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp
```

并按照屏幕上的输出操作。要手动停止脚本，请按 Ctrl – C 。

4.5 用户空间探测

为帮助调试用户空间应用程序（就像 DTrace 可做到的那样），SUSE Linux Enterprise Desktop 15 SP7 支持使用 SystemTap 进行用户空间探测。在任何用户空间应用程序中都可插入自定义探测点。因此，通过 SystemTap，您可以使用内核空间和用户空间探测来调试整个系统的行为。

要获取所需的 utrace 基础架构和 uprobes 内核模块进行用户空间探测，除了第 4.2 节 “安装和设置”中所列的软件包外，还需要安装 kernel-trace 软件包。

utrace 实施一个用于控制用户空间任务的框架。它提供了可由各种跟踪“引擎”使用的接口，该接口以可加载内核模块的形式实现。引擎会注册特定事件的回调函数，然后挂接到它们想要跟踪的任何线程。由于回调是从内核中的“安全”位置发出的，因此使得函数拥有很大的自由度，可以进行各种处理工作。通过 utrace 可以监控多个事件。例如，您可以监测系统调用进入和退出、fork() 以及正向任务发送信号等事件。有关 utrace 基础架构的更多细节，请访问 https://sourceware.org/systemtap/wiki/utrace 。

SystemTap 支持探测进入用户空间进程中的函数并从中返回、探测用户空间代码中的预定义标记，以及监控用户进程事件。

要检查当前运行的内核是否提供所需的 utrace 支持，请使用以下命令：

```
> sudo grep CONFIG_UTRACE /boot/config-`uname -r`
```

有关用户空间探测的更多细节，请访问 https://sourceware.org/systemtap/SystemTap_Beginners_Guide/userspace-probing.html。

4.6 更多信息

本章仅提供了简短的 SystemTap 概述。有关 SystemTap 的详细信息，请参考以下链接：

https://sourceware.org/systemtap/

SystemTap 项目主页。

https://sourceware.org/systemtap/wiki/

囊括了有关 SystemTap 的大量有用信息，从详细的用户和开发人员文档，到评论以及与其他工具的比较，或者常见问题和提示。另外还包含 SystemTap 脚本、示例和使用案例集合，并列出了有关 SystemTap 的最近研讨主题和论文。

https://sourceware.org/systemtap/documentation.html

提供 PDF 和 HTML 格式的 SystemTap Tutorial、SystemTap Beginner's Guide、Tapset Developer's Guide 和 SystemTap Language Reference。另外还列出了相关的手册页。

您还可以在所安装系统中的 `/usr/share/doc/packages/systemtap` 下找到 SystemTap 语言参考和 SystemTap 教程。`example` 子目录中提供了示例 SystemTap 脚本。

5 内核探测

内核探测是用于收集 Linux 内核调试信息和性能信息的一组工具。开发人员和系统管理员使用这些工具来调试内核或查找系统性能瓶颈。然后，可以使用报告的数据来微调系统，以改善性能。

您可将这些探测插入到任何内核例程，并指定在命中特定断点后要调用的处理程序。内核探测的主要优势在于，在探测中进行更改后，您无需再重构建内核并重引导系统。

要使用内核探测，通常需要编写或获取特定的内核模块。此类模块包含 **init** 和 **exit** 函数。init 函数（例如 `register_kprobe()`）可注册一个或多个探测，而 exit 函数可取消注册这些探测。注册函数定义要将探测插入到**何处**，以及在命中探测后要调用**哪个处理程序**。要一次性注册或取消注册一组探测，可以使用相关的 `register_<PROBE_TYPE>probes()` 或 `unregister_<PROBE_TYPE>probes()` 函数。

通常使用 `printk` 内核例程报告调试和状态消息。`printk` 是内核空间中与用户空间 `printf` 例程对应的例程。有关 `printk` 的详细信息，请参见 Logging kernel messages（记录内核消息）(https://www.win.tue.nl/~aeb/linux/lk/lk-2.html#ss2.8)。正常情况下，您可以通过检查 `systemd` 日记的输出（请参见《管理指南》, 第 21 章 “**`journalctl`**：查询 `systemd` 日记”）查看这些消息。有关日志文件的详细信息，请参见第 3 章 “系统日志文件”。

5.1 支持的体系结构

在下列体系结构上可**全面**实现内核探测：

- x 86
- AMD64/Intel 64
- Arm
- POWER

在下列体系结构上可**部分**实现内核探测：

- IA64（不支持对指令 `slot1` 使用探测）
- sparc64（尚未实施返回探测）

5.2 内核探测的类型

内核探测有三种类型：**Kprobes**、**Jprobes** 和 **Kretprobes**。Kretprobe 有时称作**返回探测**。您可以找到 Linux 内核中所有三种类型探测的源代码示例。请查看目录 `/usr/src/linux/samples/kprobes/`（软件包 `kernel-source`）。

5.2.1 Kprobe

Kprobe 可附加到 Linux 内核中的任何指令。注册 Kprobe 后，它会在所探测指令的第一个字节处插入一个断点。当处理器命中此断点时，将保存处理器注册，然后由 Kprobe 接管后续处理。首先执行一个**前处理程序**，然后执行所探测的指令，最后执行一个**后处理程序**。随后，控制权将传递给探测点后面的指令。

5.2.2 Jprobe

Jprobe 是通过 Kprobe 机制实施的。它将插入到函数的入口点，允许直接访问正在探测的函数的参数。其处理程序例程的参数列表与返回值必须与所探测函数相同。要结束 Jprobe，请调用 `jprobe_return()` 函数。

命中某个 jprobe 后，将保存处理器注册，并将指令指针定向到 jprobe 处理程序例程。然后，控制权将传递给与所要探测函数具有相同注册内容的处理程序。最后，该处理程序调用 `jprobe_return()` 函数，并将控制权交回给控制函数。

一般情况下，您可以在一个函数中插入多个探测。但是，Jprobe 仅限每个函数一个实例。

5.2.3 返回探测

返回探测也是通过 Kprobe 实施的。调用 `register_kretprobe()` 函数时，会将一个 kprobe 附加到所探测函数的入口。命中探测后，内核探测机制将保存所探测函数的返回地址，并调用用户定义的返回处理程序。然后，控制权将传回给所探测的函数。

在调用 `register_kretprobe()` 之前，需要设置一个 `maxactive` 参数，用于指定可以同时探测多少个函数实例。如果设置的值太小，会错过某些探测。

5.3 Kprobe API

Kprobe 的编程接口由用于注册和取消注册所有已用内核探测及关联探测处理程序的函数组成。有关这些函数及其参数的详细说明，请参见第 5.5 节 “更多信息”中的信息来源。

`register_kprobe()`
: 在指定的地址上插入一个断点。命中该断点时，将调用 `pre_handler` 和 `post_handler`。

`register_jprobe()`
: 在指定的地址中插入一个断点。该地址须是所探测函数的第一个指令的地址。命中该断点时，将运行指定的处理程序。该处理程序的参数列表和返回类型应与所探测的函数相同。

`register_kretprobe()`
: 为指定的函数插入一个返回探测。当所探测函数返回值时，将运行指定的处理程序。此函数在成功时返回 0，在失败时返回带负号的错误编号。

`unregister_kprobe()`、`unregister_jprobe()`、`unregister_kretprobe()`
: 去除指定的探测。注册探测后便可随时使用该探测。

`register_kprobes()`、`register_jprobes()`、`register_kretprobes()`
: 在指定的阵列中插入每个探测。

`unregister_kprobes()`、`unregister_jprobes()`、`unregister_kretprobes()`
: 去除指定阵列中的每个探测。

`disable_kprobe()`、`disable_jprobe()`、`disable_kretprobe()`
: 暂时禁用指定的探测。

`enable_kprobe()`、`enable_jprobe()`、`enable_kretprobe()`
: 暂时启用已禁用的探测。

5.4 debugfs 界面

在最新的 Linux 内核中，Kprobe 工具使用内核的 `debugfs` 接口。此接口可以列出所有已注册的探测并全局打开或关闭所有探测。

5.4.1 列出已注册的内核探测

/sys/kernel/debug/kprobes/list 文件中列出了当前已注册的所有探测。

```
saturn.example.com:~ # cat /sys/kernel/debug/kprobes/list
c015d71a  k  vfs_read+0x0   [DISABLED]
c011a316  j  do_fork+0x0
c03dedc5  r  tcp_v4_rcv+0x0
```

第一列列出探测所要插入到的内核中的地址。第二列列显探测的类型：k 表示 kprobe，j 表示 jprobe，r 表示返回探测。第三列指定探测的符号、偏移和可选模块名称。后面的可选列包含探测的状态信息。如果探测插入到不再有效的虚拟地址，将以 [GONE] 标记。如果探测被暂时禁用，将以 [DISABLED] 标记。

5.4.2 全局启用/禁用内核探测

/sys/kernel/debug/kprobes/enabled 文件表示一个开关，可用于全局以及强制性打开或关闭所有已注册的内核探测。要关闭这些探测，只需在命令行中输入

```
# echo "0" > /sys/kernel/debug/kprobes/enabled
```

（以 root 身份）。要再次打开这些探测，请输入

```
# echo "1" > /sys/kernel/debug/kprobes/enabled
```

此类操作不会更改探测的状态。如果某个探测被暂时禁用，在输入后一条命令之后，该探测不会自动启用，而是保持 [DISABLED] 状态。

5.5 更多信息

有关内核探测的详细信息，请查看以下信息源：

- `/usr/src/linux/Documentation/trace/kprobes.txt`（软件包 `kernel-source`）中提供了有关内核探测的综合信息，这些信息更多地以技术为主。
- `/usr/src/linux/samples/kprobes/` 目录（软件包 `kernel-source`）中提供了所有三种类型的探测（以及相关 `Makefile`）的示例。
- 在 The Linux Kernel Module Programming Guide（Linux 内核模块编程指南）（https://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html）中可以找到有关 Linux 内核模块和 `printk` 内核例程的深入信息

6 使用 Perf 进行基于硬件的性能监控

Perf 是一个用于访问处理器性能监控单元 (PMU) 以及记录和显示软件事件（例如页错误）的界面。它支持系统范围的监控、按线程的监控和 KVM 虚拟化 Guest 监控。

可以在报告中存储生成的信息。例如，此报告包含有关指令指针的信息，或线程执行的代码的信息。

Perf 由两部分组成：

- 集成到 Linux 内核的代码，会向硬件发出指令。
- `perf` 用户空间实用程序，可让您使用内核代码并帮助您分析收集的数据。

6.1 基于硬件的监控

进行性能监控意味着需要收集有关应用程序或系统执行情况的信息。可以通过基于软件的方式或者从 CPU 或芯片组获取这些信息。Perf 集成了这两种方式。

许多新式处理器都包含性能监控单元 (PMU)。PMU 的设计和功能与特定的 CPU 相关。例如，寄存器数量、支持的计数器和功能因 CPU 实现而异。

每个 PMU 模型包含一组寄存器：性能监控配置 (PMC) 和性能监控数据 (PMD)。这两个寄存器都是可读的，但只有 PMC 是可写的。这些寄存器用于存储配置信息和数据。

6.2 采样和计数

Perf 支持多种分析模式：

- **计数：** 统计某个事件的发生次数。
- **基于事件的采样：** 这是一种不太精确的统计方式：每当发生的事件数达到特定阈值时，就会记录一个样本。
- **基于时间的采样：** 这是一种不太精确的统计方式：按照定义的频率记录样本。
- **基于指令的采样（仅限 AMD64）：** 处理器跟踪按给定时间间隔出现的指令，并对这些指令生成的事件采样。这样便可以跟踪各个指令，并查看哪些是对性能至关重要的指令。

6.3 安装 Perf

Perf 内核代码已包含在默认内核中。要使用用户空间实用程序，请安装软件包 `perf`.

6.4 Perf 子命令

为了收集必要信息，**`perf`** 工具中包含多个子命令。本节概述了最常用的命令。

要以手册页的形式查看任一子命令的帮助，请使用 **`perf help`**`SUBCOMMAND` 或 **`man perf-`**`SUBCOMMAND`。

`perf stat`

启动一个程序，并创建在该程序退出后显示的统计概览。**`perf stat`** 用于统计事件。

`perf record`

启动一个程序，并创建包含性能计数器信息的报告。该报告作为 `perf.data` 存储在当前目录中。**`perf record`** 用于对事件进行采样。

`perf report`

显示先前使用 **`perf record`** 创建的报告。

`perf annotate`

显示报告文件以及已执行代码的批注版本。如果安装了调试符号，则还会显示源代码。

`perf list`

列出 Perf 可以报告的当前内核和您的 CPU 事件的类型。您可以按类别过滤事件类型。例如，要仅查看硬件事件，请使用 **`perf list hw`**。

`perf_event_open` 的手册页提供了最重要事件的简短说明。例如，要查找 `branch-misses` 事件的说明，请搜索 `BRANCH_MISSES`（请注意拼写差异）：

```
> man perf_event_open | grep -A5 BRANCH_MISSES
```

事件有时可能模糊不清。小写的硬件事件名称并非原始硬件事件的名称，而是 Perf 创建的别名的名称。这些别名对应于每个受支持处理器上名称不同但定义类似的硬件事件。例如，在 Intel 处理器上，`cpu-cycles` 事件对应于硬件事件 `UNHALTED_CORE_CYCLES`。而在 AMD 处理器上，则对应于硬件事件 `CPU_CLK_UNHALTED`。

Perf 还允许测量特定于硬件的原始事件。要查找这些事件的说明，请查看 CPU 供应商的《Architecture Software Developer's Manual》（体系结构软件开发人员手册）。第 6.7 节 “更多信息”中提供了 AMD64/Intel 64 处理器的相关文档链接。

perf top

显示发生的系统活动。

perf trace

此命令的行为与 **strace** 类似。使用此子命令可以查看特定的线程或进程执行了哪些系统调用，以及该线程或进程收到了哪些信号。

6.5 统计特定类型的事件

要统计某个事件（例如 **perf list** 显示的事件）的发生次数，请使用：

```
# perf stat -e EVENT -a
```

要一次性统计多种类型的事件，请列出这些事件并以逗号分隔。例如，要统计 `cpu-cycles` 和 `instructions`，请使用：

```
# perf stat -e cpu-cycles,instructions -a
```

要停止会话，请按 **Ctrl**–**C** 。

还可以统计某个事件在特定时间范围内发生的次数：

```
# perf stat -e EVENT -a -- sleep TIME
```

请将 `TIME` 替换为以秒为单位的值。

6.6 记录特定于特定命令的事件

可通过多种方式来对特定于特定命令的事件采样：

- 要创建新调用的命令的报告，请使用：

  ```
  # perf record COMMAND
  ```

然后正常使用启动的进程。退出该进程时，Perf 会话也会停止。

- 要在运行新调用的命令时创建整个系统的报告，请使用：

```
# perf record -a COMMAND
```

然后正常使用启动的进程。退出该进程时，Perf 会话也会停止。

- 要创建已运行的进程的报告，请使用：

```
# perf record -p PID
```

请将 PID 替换为进程 ID。要停止会话，请按 Ctrl–C 。

现在，可使用以下命令查看收集到的数据 (perf.data)：

```
> perf report
```

这会打开一个伪图形界面。要获得帮助，请按 H 。要退出，请按 Q 。

如果您偏向于使用图形界面，请尝试 Perf 的 GTK+ 界面：

```
> perf report --gtk
```

不过，GTK+ 界面提供的功能很有限。

6.7 更多信息

本章仅提供了简短概述。有关详细信息，请参考以下链接：

https://perf.wiki.kernel.org/index.php/Main_Page

项目主页。还提供了有关使用 **perf** 的教程。

https://www.brendangregg.com/perf.html

包含许多单行 **perf** 用法示例的非官方页面。

https://web.eece.maine.edu/~vweaver/projects/perf_events/

包含多个资源的非官方页面，主要与 Perf 的 Linux 内核代码及其 API 相关。例如，此页面包含 CPU 兼容性表和编程指南。

https://www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

此 Intel Architectures Software Developer's Manual, Volume 3B.

https://support.amd.com/TechDocs/24593.pdf

此 AMD Architecture Programmer's Manual, Volume 2.

第 7 章 “OProfile — 系统范围的分析器”

请参阅此章了解其他性能优化。

7 OProfile — 系统范围的分析器

OProfile 是用于动态程序分析的分析器。它可以调查运行中程序的行为并收集信息。您可以查看这些信息，并从中获得用于进一步优化的提示。

无需重新编译或使用封装程序库即可使用 OProfile。甚至不需要内核补丁。在分析应用程序时，开销应该比较低，具体取决于工作负载和采样频率。

7.1 概念概述

OProfile 由一个内核驱动程序以及一个用于收集数据的守护程序构成。它使用许多处理器上都会提供的硬件性能计数器。OProfile 能够分析所有代码，包括内核、内核模块、内核中断处理程序、系统共享库和其他应用程序。

新式处理器支持通过硬件由性能计数器执行分析。根据所用的处理器，可能会提供许多计数器，可以使用要计入的事件对每个计数器进行编程。每个计数器都有一个值来确定采样频率。值越小，则使用样本的频率越高。

在执行后处理步骤期间，将收集所有信息，并将指令地址映射到某个函数名称。

7.2 安装和要求

要使用 OProfile，请安装 `oprofile` 软件包。OProfile 可在 AMD64/Intel 64、IBM Z 和 POWER 处理器上运行。

为您要分析的相关应用程序安装 `*-debuginfo` 软件包非常有用。要分析内核，还需要安装 `debuginfo` 软件包。

7.3 可用的 OProfile 实用程序

OProfile 包含多个实用程序，用于处理分析进程及其分析的数据。以下列表简要汇总了本章中使用的程序：

opannotate

输出带批注的源代码或程序集列表（混合了分析信息）。可将带批注的报告与 **addr2line** 结合使用，以识别可能存在热点的源文件和行。有关更多信息，请参见**man addr2line**。

operf

分析器工具。例如，分析停止后，默认存储在 CUR_DIR/oprofile_data/samples/current 中的数据可由 **opreport** 处理。

ophelp

列出可用事件和简短说明。

opimport

将样本数据库文件从外部二进制格式转换为专用于平台的格式。

opreport

基于分析的数据生成报告。

7.4 使用 OProfile

使用 OProfile 可以分析内核和应用程序。分析内核时，请告知 OProfile 要在何处查找 vmlinuz* 文件。使用 --vmlinux 选项并将其指向 vmlinuz*（一般位于 /boot 中）。如果您需要分析内核模块，OProfile 默认便会执行此操作。但是，请务必查看 https://oprofile.sourceforge.net/doc/kernel-profiling.html。

大多数应用程序不需要分析内核，因此您应该使用 --no-vmlinux 选项来减少信息量。

7.4.1 创建报告

启动守护程序、收集数据、停止守护程序，并创建应用程序 COMMAND 的报告。

1. 打开外壳，并以 root 身份登录。
2. 确定是要分析还是不分析 Linux 内核：
   a. **分析 Linux 内核：** 执行以下命令，因为 **operf** 只能处理未压缩的映像：

```
> cp /boot/vmlinux-`uname -r`.gz /tmp
```

```
> gunzip /tmp/vmlinux*.gz
> operf--vmlinux=/tmp/vmlinux* COMMAND
```

b. **不分析 Linux 内核：** 使用以下命令：

```
# operf --no-vmlinux COMMAND
```

要在输出中查看哪些函数调用了其他函数，请额外使用 `--callgraph` 选项并设置最大 `DEPTH`：

```
# operf --no-vmlinux --callgraph
DEPTH COMMAND
```

3. **operf** 将其数据写入 `CUR_DIR/oprofile_data/samples/current`。**operf** 命令完成（或者通过按 `Ctrl`–`C` 中止）后，即可使用 **oreport** 分析数据：

```
# opreport
Overflow stats not available
CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt
          TIMER:0|
  samples|      %|
------------------
    84877 98.3226 no-vmlinux
...
```

7.4.2 获取事件配置

事件配置的一般过程如下：

1. 首先使用事件 `CPU-CLK_UNHALTED` 和 `INST_RETIRED` 找到优化机会。
2. 使用特定的事件查找瓶颈。要列出事件，请使用命令 **perf list**。

如果您需要分析特定的事件，请先使用 **ophelp** 命令查看处理器支持的可用事件（从 Intel Core i5 CPU 生成的示例输出）：

```
# ophelp
```

```
oprofile: available events for CPU type "Intel Architectural Perfmon"

See Intel 64 and IA-32 Architectures Software Developer's Manual
Volume 3B (Document 253669) Chapter 18 for architectural perfmon events
This is a limited set of fallback events because oprofile does not know your CPU
CPU_CLK_UNHALTED: (counter: all))
        Clock cycles when not halted (min count: 6000)
INST_RETIRED: (counter: all))
        number of instructions retired (min count: 6000)
LLC_MISSES: (counter: all))
        Last level cache demand requests from this core that missed the LLC (min
 count: 6000)
        Unit masks (default 0x41)
        ----------
        0x41: No unit mask
LLC_REFS: (counter: all))
        Last level cache demand requests from this core (min count: 6000)
        Unit masks (default 0x4f)
        ----------
        0x4f: No unit mask
BR_MISS_PRED_RETIRED: (counter: all))
        number of mispredicted branches retired (precise) (min count: 500)
```

使用选项 `--event` 指定性能计数器事件。可以使用多个选项。此选项需要事件名称（通过 **`ophelp`** 获得）和采样率，例如：

```
# operf --events CPU_CLK_UNHALTED:100000
```

警告：使用 CPU_CLK_UNHALTED 设置采样率

设置较低的采样率可能会严重影响系统性能，而设置较高的采样率可能会给系统造成较大程度的干扰，导致数据无用。建议微调要使用和不使用 OProfile 监控的性能指标，并通过试验来确定对性能产生的干扰最轻的最小采样率。

7.5 生成报告

在生成报告之前，请确保 **operf** 已停止。除非您已使用 `--session-dir` 提供了输出目录，否则 **operf** 会将其数据写入 `CUR_DIR`/oprofile_data/samples/current，并且报告工具 **opreport** 和 **opannotate** 默认会在此位置查找数据。

不使用任何选项调用 **opreport** 可以提供完整摘要。使用某个可执行文件作为参数，检索仅来自于此可执行文件的分析数据。如果您要分析以 C++ 编写的应用程序，请使用 `--demangle smart` 选项。

opannotate 生成的输出包含源代码中的批注。结合使用以下选项运行此命令：

```
# opannotate --source \
   --base-dirs=BASEDIR \
   --search-dirs=SEARCHDIR \
   --output-dir=annotated/ \
   /lib/libfoo.so
```

选项 `--base-dir` 包含从调试源文件拆分出的路径逗号分隔列表。在 `--search-dirs` 中查找前已搜索过这些路径。`--search-dirs` 选项也是一个用于搜索源文件的目录逗号分隔列表。

注意：带批注源代码中的误差

由于编译器优化功能，代码可能会消失或显示在不同的位置。参考 https://oprofile.sourceforge.net/doc/debug-info.html 中的信息可以全面了解这种行为的含义。

7.6 更多信息

本章仅提供了简短概述。有关详细信息，请参考以下链接：

https://oprofile.sourceforge.net

项目主页。

手册页

有关不同工具的选项的详细说明。

`/usr/share/doc/packages/oprofile/oprofile.html`

包含 OProfile 手册。

https://developer.intel.com/

Intel 处理器的体系结构参考。

8 动态调试 - 内核调试消息

动态调试是 Linux 内核中的一项强大的调试功能，它允许您在运行时启用和禁用调试消息，而无需重新编译内核或重引导系统。

您可以在以下几种情况下使用动态调试，例如：

- 内核问题查错
- 为新硬件开发驱动程序
- 跟踪和审计安全事件

8.1 动态调试的优势

下面列出了动态调试的某些优势：

实时调试

动态调试支持在不重引导系统的情况下调试消息。这种实时功能对于诊断生产环境中的问题至关重要。

选择性调试

您可以为内核的特定部分甚至单个模块启用调试消息，从而将重点放在相关信息上。

性能微调

使用动态调试来监控和优化内核性能，只需根据当前的分析要求选择性地启用或禁用调试消息。

8.2 检查动态调试的状态

默认安装的受支持内核版本已内置动态调试。要检查动态调试的状态，请以 root 用户身份运行以下命令：

```
# zcat /proc/config.gz | grep CONFIG_DYNAMIC_DEBUG
```

如果动态调试已编译到内核中，您应该看到类似于以下内容的输出：

```
CONFIG_DYNAMIC_DEBUG=y
CONFIG_DYNAMIC_DEBUG_CORE=y
```

8.3 使用动态调试

要在正在运行的内核中启用特定的调试消息或日志，您可以使用 **echo** 命令向 /sys/kernel/debug/dynamic_debug/control 文件写入数据。

以下示例说明了动态调试的一些简单用法：

注意

动态调试依赖于内核代码中所嵌入的特定调试宏，例如 pr_debug。内核开发人员使用这些宏将调试消息插入到代码中。

本节中的示例假设 pr_debug 宏正常工作，因为允许对正在运行的内核进行动态调试。

为特定内核源代码文件启用调试消息

要为特定内核源代码文件启用调试消息，请使用以下示例：

```
# echo "file FILE_NAME.c +p" > /sys/kernel/debug/dynamic_debug/control
```

为特定内核模块启用调试消息

要为特定内核模块启用调试消息，请使用以下示例：

```
# echo "module MODULE_NAME +p" > /sys/kernel/debug/dynamic_debug/control
```

禁用调试消息

要禁用以前为特定内核源代码文件或内核模块启用的调试消息，请运行带有 **-p** 选项的 **echo** 命令。例如：

```
# echo "file FILE_NAME.c -p" > /sys/kernel/debug/dynamic_debug/control
```

```
# echo "module MODULE_NAME -p" > /sys/kernel/debug/dynamic_debug/control
```

有关动态调试及其使用场景的详细信息，请参见其官方文档 (https://www.kernel.org/doc/html/latest/admin-guide/dynamic-debug-howto.html)。

8.4 查看动态调试消息

您可以运行 **dmesg** 并通过 **grep** 过滤输出，来查看根据启用的配置生成的动态调试消息。例如：

```
# dmesg | grep -i "FILE_NAME.c"
```

或者，要在系统消息生成时持续监控系统消息，可以使用带有 **-f** 选项的 **tail** 命令：

```
# tail -f /var/log/messages
```

IV 资源管理

9 一般系统资源管理 105

10 内核控制组 110

11 自动平衡非一致性内存访问 (NUMA) 119

12 电源管理 124

9 一般系统资源管理

微调系统不仅能优化内核或发挥应用程序的最大作用，还能从一开始就设置一个精简且快速的系统。设置分区和文件系统的方式可能会影响服务器的速度。活动服务的数目以及安排例行任务的方式也会影响性能。

9.1 规划安装

精心规划的安装可确保完全按照您的需求和目的设置系统。另外，在微调系统时还可以节省大量时间。本节建议的所有更改都可以在安装期间的安装设置步骤中进行。有关详细信息，请参见《Deployment Guide》, 第 5 章 “Installation steps”, 第 5.14 节 “Installation settings”。

9.1.1 分区

根据服务器的应用程序范围和硬件布局，分区方案可能会影响计算机的性能（不过影响程度较小）。本手册不会建议如何为特定工作负载选择不同分区方案，因为这超出了本手册的讨论范围。不过，以下原则能够对性能产生积极影响。使用外部存储系统时，这些原则不适用。

- 确保磁盘上始终留有一定数量的可用空间，因为如果磁盘已满，则无法提供最佳性能。
- 例如，可通过以下方式将同时进行的读取和写入访问分散到不同的磁盘：
    - 为操作系统、数据和日志文件使用不同的磁盘
    - 将邮件服务器的假脱机目录放在单独的磁盘上
    - 将主服务器的用户目录在不同的磁盘之间分发

9.1.2 安装范围

虽然安装范围不会直接影响计算机的性能，但认真选择软件包范围还是有益处的。建议安装满足运行服务器所需的最少量的软件包。包含最少量软件包的系统更容易维护，且潜在安全问题也更少。此外，定制安装范围还可确保默认不会启动不必要的服务。

SUSE Linux Enterprise Desktop 允许您在“安装摘要”屏幕上自定义安装范围。默认情况下，您可以选择或去除特定任务的预配置软件集，但也可以启动 YaST 软件管理器以基于软件包进行精细选择。

不一定所有情况都需要以下一个或多个默认软件集：

GNOME 桌面环境

服务器极少需要完整的桌面环境。如果需要图形环境，较为经济实惠的解决方案（例如 IceWM）便已足够。

X Window 系统

如果只通过命令行管理服务器及其应用程序，建议不要安装此软件集。但请注意，从远程计算机运行 GUI 应用程序需要此软件集。如果您的应用程序是通过 GUI 管理的，或者您偏好 GUI 版本的 YaST，请保留此软件集。

打印服务器

仅当需要从计算机打印时，才需要此软件集。

9.1.3 默认目标

运行 X Window 系统会消耗大量资源，在服务器上极少需要它。强烈建议在目标 `multi-user.target` 中启动该系统。您仍可以远程启动图形应用程序。

9.2 禁用不必要的服务

默认安装会启动多个服务（数量视安装范围而定）。由于每个服务都会消耗资源，因此建议禁用不需要的服务。运行 YaST › 系统 › 服务管理器启动服务管理模块。

如果您使用图形版本的 YaST，可以单击列标题对服务列表进行排序。使用此方法可以获取当前正在运行服务的概览。单击启动/停止按钮禁用运行中会话的服务。要永久禁用该服务，请单击启用/禁用按钮。

以下列表显示了在安装 SUSE Linux Enterprise Desktop 后默认启动的服务。检查您需要哪些组件，然后禁用其他组件：

alsasound

加载 Linux 高级声音系统。

auditd

Audit 系统的守护程序（有关细节，请参见《安全和强化指南》）。如果您不使用 Audit，请禁用此组件。

bluez-coldplug

处理蓝牙硬件保护装置的冷插入。

cups

打印机守护程序。

java.binfmt_misc

启用 `*.class` 或 `*.jar` Java 程序的执行。

nfs

挂载 NFS 时所需的服务。

smbfs

从 Windows* 服务器挂载 SMB/CIFS 文件系统时所需的服务。

splash/splash_early

在启动时显示欢迎屏幕。

9.3 文件系统和磁盘访问

硬盘是计算机系统中速度最慢的组件，因此经常造成瓶颈。使用最适合您工作负载的文件系统有助于提升性能。使用特殊挂载选项或指定进程的 I/O 优先级可以进一步加速系统。

9.3.1 文件系统

SUSE Linux Enterprise Desktop 随附了多个文件系统，包括 Btrfs、Ext4、Ext3、Ext2 和 XFS。每个文件系统都有各自的优点和缺点。

9.3.1.1 NFS

NFS 操作指南中详细介绍了 NFS（版本 3）微调方法，其网址为：https://nfs.sourceforge.net/nfs-howto/。挂载 NFS 共享时，先试验一下通过挂载选项 `wsize` 和 `rsize` 将读写块的大小增加到 32768。

9.3.2 时戳更新策略

文件系统中的每个文件和目录都有三个关联的时戳：上次读取文件的时间（称为**访问时间**）、上次修改文件数据的时间（称为**修改时间**），以及上次修改文件元数据的时间（称为**更改时间**）。如果让访问时间始终保持最新，会产生极大的性能开销，因为每次进行只读访问都会产生一个写入操作。默认情况下，只会在当前文件访问时间超过一天或者早于文件修改时间或更改时间时，每个文件系统才会更新访问时间。此特性称为**相对访问时间**，对应的挂载选项为 `relatime`。您可以使用 `noatime` 挂载选项禁用访问时间更新，但需要校验应用程序确实不使用这种更新。文件服务器和 Web 服务器或网络存储系统可能都需要如此。如果默认相对访问时间更新策略不适合您的应用程序，请使用 `strictatime` 挂载选项。

某些文件系统（例如 Ext4）还支持迟缓时戳更新。使用 `lazytime` 挂载选项启用此功能后，所有时戳的更新将在内存中发生，但不会写入到磁盘。只有在响应 `fsync` 或 `sync` 系统调用时、由于其他原因（例如文件大小更新）而写入文件信息时、当时戳超过 24 小时时，或者需要从内存中逐出已缓存的文件信息时，才发生这种更新。

要更新用于文件系统的挂载选项，请直接编辑 `/etc/fstab`，或者在通过 YaST 分区程序编辑或添加分区时使用 Fstab 选项对话框。

9.3.3 使用 `ionice` 指定磁盘访问优先级

`ionice` 命令可让您指定单个进程的磁盘访问优先级。这样，您便可以降低磁盘访问量大但时间不紧迫的后台进程（例如备份作业）的 I/O 优先级。**`ionice`** 还可用于提高特定进程的 I/O 优先级，以确保该进程始终可以立即访问磁盘。使用此功能时需要注意，标准写入操作是缓存在页缓存中，以后只能由独立的内核进程写回到永久存储设备。因此，I/O 优先级设置通常不适用于这些写入操作。另请注意，只有 blk-mq I/O 路径的 **BFQ** I/O 调度程序才遵守 I/O 类和优先级设置（请参见第 13.2 节 “使用 blk-mq I/O 路径的可用 I/O 电梯算法”）。您可以设置以下三个调度类：

空闲

仅当没有其他进程请求磁盘 I/O 时，才会向空闲调度类中的进程授予磁盘访问权限。

尽力而为

未请求特定 I/O 优先级的任何进程都会使用的默认调度类。可在从 `0` 到 `7` 的范围内（`0` 是最高优先级）调整此类中的优先级。对于以相同的尽力而为 (best-effort) 优先级运行的程序将按循环方式提供服务。某些内核版本会按不同的方式处理尽力而为 (best-effort) 类中的优先级 — 有关细节，请参见 `ionice(1)` 手册页。

实时

始终最先为此类中的进程授予磁盘访问权限。可在从 `0` 到 `7` 的范围内（`0` 是最高优先级）微调优先级。请慎用此类，因为它可能会导致其他进程无法获得访问权限。

有关更多细节和确切的命令语法，请参见 `ionice(1)` 手册页。如果您需要更可靠地控制每个应用程序可用的带宽，请使用第 10 章 “内核控制组”中所述的内核控制组。

10 内核控制组

内核控制组 (“cgroup”) 是一种内核功能，可用于分配和限制进程的硬件与系统资源。进程也可采用层次树状结构的形式进行整理。

10.1 概述

每个进程只会分配到一个管理 cgroup。Cgroup 按层次树状结构排序。您可针对单个进程或针对层次树中的所有分支设置资源限制，例如 CPU、内存、磁盘 I/O 或网络带宽用量。

在 SUSE Linux Enterprise Desktop 上，`systemd` 使用 cgroup 以分组（`systemd` 称之为切片）形式整理所有进程。`systemd` 还提供了用于设置 cgroup 属性的接口。

命令 **`systemd-cgls`** 显示层次树状结构。

内核 cgroup API 有两个版本：v1 和 v2。此外，可能有多个 cgroup 层次结构公开不同的 API。从众多可能的组合来看，有两种可行的选择：

- 统一：包含控制器的 v2 层次结构
- 混合：不包含控制器的 v2 层次结构；控制器位于 v1 层次结构中（已弃用）

默认模式为统一。有种混合模式可以为需要 v1 的应用程序提供向后兼容性。

您只能设置一种模式。

10.1.1 混合 cgroup 层次结构

注意：弃用注意事项

cgroup v1 已被弃用，在将来的版本中可能会将其去除。

要启用混合控制组层次结构，请将 `systemd.unified_cgroup_hierarchy=0` 作为内核命令行参数追加到 GRUB 2 引导加载程序。有关配置 GRUB 2 的更多细节，请参见《管理指南》，第 18 章 “引导加载程序 GRUB 2”。

10.2 资源统计

可以通过将进程组织到不同的 cgroup 来获取每个 cgroup 的资源消耗数据。

统计进程较小，但开销并非为零，它所造成的影响取决于工作负载。对一个单元激活统计也会对同一切片中的所有单元、该切片的所有父切片，以及包含在这些父切片中的单元隐式激活统计。

可以使用诸如 `MemoryAccounting=` 的指令为每个单元设置统计，或者在 `/etc/systemd/system.conf` 中使用 `DefaultMemoryAccounting=` 指令为所有单元全局设置统计。有关可用指令的详尽列表，请参见 **`man systemd.resource-control`**。

10.3 设置资源限制

注意：隐式资源消耗

请注意，资源消耗隐式取决于工作负载的执行环境（例如，库/内核中数据结构的大小、实用程序的派生行为、计算效率）。因此，如果环境发生变化，建议您（重新）校准您的限制。

可以使用 **`systemctl set-property`** 命令设置 cgroup 的限制。语法是：

```
# systemctl set-property [--runtime] NAME PROPERTY1=VALUE [PROPERTY2=VALUE]
```

配置的值会立即应用。或者，也使用 `--runtime` 选项，使新值在重引导后不会保留。

请将 `NAME` 替换为 `systemd` 服务、范围或切片名称。

有关完整的属性列表和更多细节，请参见 **`man systemd.resource-control`**。

10.4 使用 TasksMax 防止派生炸弹

`systemd` 支持为每个叶单元配置任务计数限制，或配置各个切片的聚合限制。上游 `systemd` 随附了用于限制每个单元中的任务数量的默认值（内核全局限制的 15%，运行 **`/usr/sbin/sysctl kernel.pid_max`** 可查看总限制）。每个用户的切片限制为内核限制的 33%。但对于 SUSE Linux Enterprise Desktop，情况并非如此。

10.4.1 查找当前的默认 TasksMax 值

在实际应用中，显然不存在一个适用于所有使用场景的默认设置。SUSE Linux Enterprise Desktop 随附了两种自定义配置，它们会覆盖上游系统单元和用户切片的默认值，并将这两者都设置为 `infinity`。`/usr/lib/systemd/system.conf.d/__25-defaults-SLE.conf` 包含以下行：

```
[Manager]
DefaultTasksMax=infinity
```

`/usr/lib/systemd/system/user-.slice.d/25-defaults-SLE.conf` 包含以下行：

```
[Slice]
TasksMax=infinity
```

使用 **`systemctl`** 校验 DefaultTasksMax 值：

```
> systemctl show --property DefaultTasksMax
DefaultTasksMax=infinity
```

`infinity` 表示没有限制。并不要求一定要更改默认值，但设置某些限制可能有助于防止失控进程导致系统崩溃。

10.4.2 覆盖 DefaultTasksMax 值

通过创建新的覆盖文件 `/etc/systemd/system.conf.d/90-system-tasksmax.conf` 更改全局 `DefaultTasksMax` 值，并写入下面几行以便为每个系统单元设置新的默认任务数限制（256 个）：

```
[Manager]
DefaultTasksMax=256
```

加载新设置，然后校验设置是否已更改：

```
> sudo systemctl daemon-reload
> systemctl show --property DefaultTasksMax
DefaultTasksMax=256
```

根据您的需要调整此默认值。可根据需要在单个服务上设置不同的限制。本示例适用于 MariaDB。首先检查当前活动值：

```
> systemctl status mariadb.service
  ● mariadb.service - MariaDB database server
    Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor
 preset>
    Active: active (running) since Tue 2020-05-26 14:15:03 PDT; 27min ago
      Docs: man:mysqld(8)
            https://mariadb.com/kb/en/library/systemd/
 Main PID: 11845 (mysqld)
    Status: "Taking your SQL requests now..."
     Tasks: 30 (limit: 256)
    CGroup: /system.slice/mariadb.service
            └─11845 /usr/sbin/mysqld --defaults-file=/etc/my.cnf --user=mysql
```

Tasks 行显示 MariaDB 中当前有 30 个任务正在运行，而默认上限为 256，这对于数据库而言并不足够。以下示例演示如何将 MariaDB 的限制提高至 8192。

```
> sudo systemctl set-property mariadb.service TasksMax=8192
> systemctl status mariadb.service
● mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor
 preset: disab>
  Drop-In: /etc/systemd/system/mariadb.service.d
           └─50-TasksMax.conf
   Active: active (running) since Tue 2020-06-02 17:57:48 PDT; 7min ago
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
  Process: 3446 ExecStartPre=/usr/lib/mysql/mysql-systemd-helper upgrade
 (code=exited, sta>
  Process: 3440 ExecStartPre=/usr/lib/mysql/mysql-systemd-helper install
 (code=exited, sta>
 Main PID: 3452 (mysqld)
   Status: "Taking your SQL requests now..."
    Tasks: 30 (limit: 8192)
   CGroup: /system.slice/mariadb.service
           └─3452 /usr/sbin/mysqld --defaults-file=/etc/my.cnf --user=mysql
```

`systemctl set-property` 应用新限制，并创建一个可持久保存的插入式文件 `/etc/systemd/system/mariadb.service.d/50-TasksMax.conf`，其中仅包含您要应用于现有单元文件的更改。值不一定必须是 8192，但应该是适合您工作负载的限制。

10.4.3 针对用户的默认 TasksMax 限制

针对用户的默认限制应较高，因为用户会话需要更多的资源。通过创建新文件（例如 `/etc/systemd/system/user-.slice.d/40-user-taskmask.conf`）为任何用户设置您自己的默认值。以下示例设置的默认值为 16284：

```
[Slice]
TasksMax=16284
```

注意：数字前缀参考

请参见 《管理指南》, 第 19 章 “systemd 守护程序”, 第 19.5.3 节 “手动创建插入式文件” 来了解插入式文件所需的数字前缀。

然后重新加载 systemd 以加载新值，并校验更改：

```
> sudo systemctl daemon-reload
> systemctl show --property TasksMax user-1000.slice
TasksMax=16284
```

如何知道要使用哪些值？这因您的工作负载、系统资源和其他资源配置而异。如果 `TasksMax` 值太小，您可能会看到诸如**无法派生（资源暂时不可用）**、**无法创建用于处理新连接的线程**，以及**错误：函数调用 “fork” 失败并返回错误代码 11 “资源暂时不可用”**等错误消息。

有关在 systemd 中配置系统资源的详细信息，请参见 `systemd.resource-control (5)`。

10.5 使用 cgroup 进行 I/O 控制

本节介绍如何使用 Linux 内核的块 I/O 控制器来指定 I/O 操作的优先级或限制此类操作。本节将利用 systemd 提供的方法来配置 cgroup，并讨论在处理按比例 I/O 控制时可能存在的陷阱。

10.5.1 先决条件

以下小节介绍了在设计和配置系统时必须提前采取的措施，因为在运行时期间无法更改这方面的设置。

10.5.1.1 文件系统

应该使用 cgroup 写回感知的文件系统（否则无法计算写回费用）。建议的 SUSE Linux Enterprise Desktop 文件系统增加了对下述上游版本的支持：

- Btrfs (v4.3)
- Ext4 (v4.3)
- XFS (v5.3)

从 SUSE Linux Enterprise Desktop 15 SP3 开始，可以使用任何命名文件系统。

10.5.1.2 块 I/O 调度程序

限制策略在堆栈中的更高级别实施，因此不需要进行任何额外的调整。按比例 I/O 控制策略有两种不同的实施方式：BFQ 控制器和基于成本的模型。本节介绍 BFQ 控制器。要对特定的设备按比例实施该策略，我们必须确保 BFQ 是所选的调度程序。检查当前调度程序：

```
> cat /sys/class/block/sda/queue/scheduler
mq-deadline kyber bfq [none]
```

将调度程序切换到 BFQ：

```
# echo bfq > /sys/class/block/sda/queue/scheduler
```

必须指定磁盘设备（而不是分区）。设置此属性的最佳方式是创建特定于设备的 udev 规则。SUSE Linux Enterprise Desktop 随附的 udev 规则已经为旋转式磁盘驱动器启用了 BFQ。

10.5.1.3 Cgroup 层次结构布局

一般情况下，所有任务都驻留在根 cgroup 中，它们会相互竞争。将任务分发到 cgroup 树中时，竞争只在同级 cgroup 之间发生。这一点适用于按比例 I/O 控制；限制会按层次结构聚合所有后代的吞吐量（参见下图）。

```
r
```

```
`-  a      IOWeight=100
    `- [c] IOWeight=300
    `-  d  IOWeight=100
`- [b]     IOWeight=200
```

I/O 仅源自 cgroup c 和 b。即使 c 的权重更高，也会将其视为优先级更低，因为它与 b 之间存在级别竞争。

10.5.2 配置控制数量

可将这些值永久应用于（长时间运行的）服务。

```
> sudo systemctl set-property fast.service IOWeight=400
> sudo systemctl set-property slow.service IOWeight=50
> sudo systemctl set-property throttled.service IOReadBandwidthMax="/dev/sda 1M"
```

或者，可将 I/O 控制应用于单个命令，例如：

```
> sudo systemd-run --scope -p IOWeight=400 high_prioritized_command
> sudo systemd-run --scope -p IOWeight=50 low_prioritized_command
> sudo systemd-run --scope -p IOReadBandwidthMax="/dev/sda 1M" dd if=/dev/sda
 of=/dev/null bs=1M count=10
```

10.5.3 I/O 控制行为和设置预期

以下列表项目描述了 I/O 控制行为，以及不同条件下的预期结果。

- I/O 控制对直接 I/O 操作（绕过页缓存）的效果最佳，而在实际 I/O 与调用方解耦的情况下（通常是通过页缓存进行写回），可能会出现各种不同的表现。例如，I/O 控制可能延迟生效，甚至完全无法观测到（试想小型突发操作或相互竞争的工作负载恰好从未“相遇”- 即不会同时提交 I/O，且带宽处于饱和状态）。因此，最终的 I/O 吞吐量比率并不会严格遵循配置的权重比率。
- systemd 执行配置权重的缩放（以根据较窄的 BFQ 权重范围进行调整），因此最终的吞吐量比率也不相同。

- 除了全局 sysctl 句柄之外，写回活动还取决于脏页的数量（`vm.dirty_background_ratio` 和 `vm.dirty_ratio`）。当脏页限制分布在 cgroup 之间时，各个 cgroup 的内存限制就会发挥作用，从而可能影响相关 cgroup 的 I/O 强度。
- 并非所有存储区都是相同的。I/O 控制发生在 I/O 调度程序层，这会对堆叠在不执行实际调度的存储区上的设备的设置产生影响。以跨越多个物理设备、MD RAID 甚至 Btrfs RAID 的设备映射器逻辑卷为例。对此类设置进行 I/O 控制可能颇有难度。
- 不存在可用于对读取和写入按比例控制 I/O 的单个设置。
- 按比例 I/O 控制只是能够彼此交互的策略之一（但负责任的资源设计可能会避免这种交互）。
- I/O 设备带宽不是 I/O 路径中唯一的共享资源。这涉及到全局文件系统结构，当 I/O 控制旨在保证特定的带宽时，就要考虑到此因素；但这种控制不能保证带宽，甚至可能导致优先级反转（优先的 cgroup 会等待较慢 cgroup 的事务）。
- 到目前为止，我们一直都在讨论文件系统数据的显式 I/O，但换入和换出也可以控制。不过，如果出现这种需求，系统会指出内存置备不当（或内存限制）。

10.5.4 用户会话中的资源控制

为了在用户会话中应用 cgroup 资源控制，必须将控制器委派给 `systemd` 的用户实例。SUSE Linux Enterprise Desktop 随附的 `systemd` 默认配置**不**委派任何控制器。

您可以使用插入式文件来更改委托控制器集。例如，`/etc/systemd/system/user@.service.d/60-delegate.conf` 会向所有用户添加控制器，而 `/etc/systemd/system/user@uid.service.d/60-delegate.conf` 仅向特定用户添加控制器。文件的内容应如下所示：

```
[Service]
Delegate=pids memory
```

必须通知 `systemd` 实例和受影响的用户实例，以重新加载新配置。

```
> sudo systemctl daemon-reload
> systemctl --user daemon-reexec
```

或者，受影响的用户可以注销后再登录，而不是应用第二行来重新启动其用户实例。

10.6 更多信息

- 内核文档（软件包 `kernel-source`）：`/usr/src/linux/Documentation/admin-guide/cgroup-v1` 中的文件，以及文件 `/usr/src/linux/Documentation/admin-guide/cgroup-v2.rst`。
- **`man systemd.resource-control`**
- https://lwn.net/Articles/604609/ —Brown, Neil：Control Groups Series（控制组系列，发布于 2014 年，包含 7 个部分）。
- https://lwn.net/Articles/243795/ —Corbet, Jonathan：Controlling memory use in containers（控制容器中的内存使用，发布于 2007 年）。
- https://lwn.net/Articles/236038/ —Corbet, Jonathan：Process containers（进程容器，发布于 2007 年）。

11 自动平衡非一致性内存访问 (NUMA)

需要多个 CPU 和大量内存时，硬件会存在一些物理限制。在本章中，重要的限制是 CPU 与内存之间的通讯带宽受限。非一致性内存访问 (NUMA) 便是为了解决此限制而引入的一项体系结构修改。

此配置中存在多个节点。每个节点包含所有 CPU 和内存的子集。访问主内存的速度由内存相对于 CPU 的位置决定。工作负载的性能取决于访问数据（位于执行线程的 CPU 本地）的应用程序线程。自动平衡 NUMA 可以按需将数据迁移到位于访问该数据的 CPU 本地的内存节点。使用 NUMA 硬件时，这可能会大幅提升性能（具体由工作负载而定）。

11.1 实现

自动平衡 NUMA 通过三个基本步骤完成：

1. 任务扫描程序会定期扫描任务的部分地址空间，并标记内存，以在下一次访问数据时强制引发页错误。
2. 下一次访问数据会导致 NUMA Hinting 错误。基于此错误，可将数据迁移到与访问内存的任务相关联的内存节点。
3. 要将某个任务、该任务使用的 CPU 以及该任务访问的内存保留在一起，调度程序会将共享数据的任务分到一组。

数据取消映射和页错误处理会产生开销。不过，一般情况下，访问与 CPU 关联的数据的线程会抵消这项开销。

11.2 配置

建议通过静态配置来微调 NUMA 硬件上的工作负载。为此，可以使用 **`numactl`**、**`taskset`** 或 **`cpusets`** 设置内存策略。NUMA 感知应用程序可以使用特殊 API。如果已创建静态策略，则应禁用自动平衡 NUMA，因为此时数据访问就在本地进行。

numactl --hardware 会显示计算机的内存配置，以及计算机是否支持 NUMA。下面是在一台 4 节点计算机上运行该命令后的输出。

```
> numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36 40 44
node 0 size: 16068 MB
node 0 free: 15909 MB
node 1 cpus: 1 5 9 13 17 21 25 29 33 37 41 45
node 1 size: 16157 MB
node 1 free: 15948 MB
node 2 cpus: 2 6 10 14 18 22 26 30 34 38 42 46
node 2 size: 16157 MB
node 2 free: 15981 MB
node 3 cpus: 3 7 11 15 19 23 27 31 35 39 43 47
node 3 size: 16157 MB
node 3 free: 16028 MB
node distances:
node   0   1   2   3
  0:  10  20  20  20
  1:  20  10  20  20
  2:  20  20  10  20
  3:  20  20  20  10
```

可以通过将 1 或 0 写入 /proc/sys/kernel/numa_balancing，分别为当前会话启用或禁用自动平衡 NUMA 功能。要永久启用或禁用此功能，请使用内核命令行选项 numa_balancing=[enable|disable]。

如果已启用自动平衡 NUMA，则可以配置任务扫描程序行为。任务扫描程序可以在自动平衡 NUMA 的开销与它识别最佳数据位置所需的时间之间进行平衡。

numa_balancing_scan_delay_ms
: 在扫描数据之前线程必须消耗的 CPU 时间。这可以防止由于进程生存期较短而产生开销。

numa_balancing_scan_period_min_ms**和**numa_balancing_scan_period_max_ms
: 控制扫描任务数据的频率。根据错误所在的位置，扫描频率会增大或减小。这些设置控制最小和最大扫描频率。

`numa_balancing_scan_size_mb`

控制当任务扫描程序处于活动状态时要扫描的地址空间大小。

11.3 监控

最重要的任务是为工作负载指定指标，并在启用和禁用自动平衡 NUMA 功能的情况下测量性能，以衡量相应影响。可以使用分析工具来监控本地和远程内存访问（如果 CPU 支持这种监控）。可通过 `/proc/vmstat` 中的以下参数监控自动平衡 NUMA 活动：

`numa_pte_updates`

标记 NUMA 删格化错误的基页的数量。

`numa_huge_pte_updates`

标记 NUMA 删格化错误的透明大页的数量。与 `numa_pte_updates` 结合使用可以计算标记的地址空间总大小。

`numa_hint_faults`

记录已捕获到的 NUMA 删格化错误数。

`numa_hint_faults_local`

显示本地节点上出现的删格化错误数。与 `numa_hint_faults` 结合使用可以计算本地错误与远程错误所占的百分比。如果本地删格化错误占比较高，表示工作负载更接近聚合。

`numa_pages_migrated`

记录有多少页由于位置不当而被迁移。由于迁移是一个复制操作，它占据了平衡 NUMA 所产生开销的最大一部分。

11.4 影响

下面演示了在未对内存策略进行静态微调的情况下，在一台运行 SpecJBB 2005 并使用单个 JVM 实例的 4 节点 NUMA 计算机上完成的简单测试案例。不过，这对不同工作负载产生的影响是不同的，并且此示例基于 SUSE Linux Enterprise Desktop 12 预发行版。

```
           Balancing disabled      Balancing enabled
TPut 1     26629.00 (  0.00%)     26507.00 ( -0.46%)
TPut 2     55841.00 (  0.00%)     53592.00 ( -4.03%)
```

```
TPut 3      86078.00 (  0.00%)     86443.00 (  0.42%)
TPut 4     116764.00 (  0.00%)    113272.00 ( -2.99%)
TPut 5     143916.00 (  0.00%)    141581.00 ( -1.62%)
TPut 6     166854.00 (  0.00%)    166706.00 ( -0.09%)
TPut 7     195992.00 (  0.00%)    192481.00 ( -1.79%)
TPut 8     222045.00 (  0.00%)    227143.00 (  2.30%)
TPut 9     248872.00 (  0.00%)    250123.00 (  0.50%)
TPut 10    270934.00 (  0.00%)    279314.00 (  3.09%)
TPut 11    297217.00 (  0.00%)    301878.00 (  1.57%)
TPut 12    311021.00 (  0.00%)    326048.00 (  4.83%)
TPut 13    324145.00 (  0.00%)    346855.00 (  7.01%)
TPut 14    345973.00 (  0.00%)    378741.00 (  9.47%)
TPut 15    354199.00 (  0.00%)    394268.00 ( 11.31%)
TPut 16    378016.00 (  0.00%)    426782.00 ( 12.90%)
TPut 17    392553.00 (  0.00%)    437772.00 ( 11.52%)
TPut 18    396630.00 (  0.00%)    456715.00 ( 15.15%)
TPut 19    399114.00 (  0.00%)    484020.00 ( 21.27%)
TPut 20    413907.00 (  0.00%)    493618.00 ( 19.26%)
TPut 21    413173.00 (  0.00%)    510386.00 ( 23.53%)
TPut 22    420256.00 (  0.00%)    521016.00 ( 23.98%)
TPut 23    425581.00 (  0.00%)    536214.00 ( 26.00%)
TPut 24    429052.00 (  0.00%)    532469.00 ( 24.10%)
TPut 25    426127.00 (  0.00%)    526548.00 ( 23.57%)
TPut 26    422428.00 (  0.00%)    531994.00 ( 25.94%)
TPut 27    424378.00 (  0.00%)    488340.00 ( 15.07%)
TPut 28    419338.00 (  0.00%)    543016.00 ( 29.49%)
TPut 29    403347.00 (  0.00%)    529178.00 ( 31.20%)
TPut 30    408681.00 (  0.00%)    510621.00 ( 24.94%)
TPut 31    406496.00 (  0.00%)    499781.00 ( 22.95%)
TPut 32    404931.00 (  0.00%)    502313.00 ( 24.05%)
TPut 33    397353.00 (  0.00%)    522418.00 ( 31.47%)
TPut 34    382271.00 (  0.00%)    491989.00 ( 28.70%)
TPut 35    388965.00 (  0.00%)    493012.00 ( 26.75%)
TPut 36    374702.00 (  0.00%)    502677.00 ( 34.15%)
TPut 37    367578.00 (  0.00%)    500588.00 ( 36.19%)
TPut 38    367121.00 (  0.00%)    496977.00 ( 35.37%)
TPut 39    355956.00 (  0.00%)    489430.00 ( 37.50%)
TPut 40    350855.00 (  0.00%)    487802.00 ( 39.03%)
```

```
TPut 41    345001.00 (  0.00%)    468021.00 ( 35.66%)
TPut 42    336177.00 (  0.00%)    462260.00 ( 37.50%)
TPut 43    329169.00 (  0.00%)    467906.00 ( 42.15%)
TPut 44    329475.00 (  0.00%)    470784.00 ( 42.89%)
TPut 45    323845.00 (  0.00%)    450739.00 ( 39.18%)
TPut 46    323878.00 (  0.00%)    435457.00 ( 34.45%)
TPut 47    310524.00 (  0.00%)    403914.00 ( 30.07%)
TPut 48    311843.00 (  0.00%)    459017.00 ( 47.19%)

                        Balancing Disabled        Balancing Enabled
 Expctd Warehouse          48.00 (  0.00%)          48.00 (  0.00%)
 Expctd Peak Bops      310524.00 (  0.00%)      403914.00 ( 30.07%)
 Actual Warehouse          25.00 (  0.00%)          29.00 ( 16.00%)
 Actual Peak Bops      429052.00 (  0.00%)      543016.00 ( 26.56%)
 SpecJBB Bops            6364.00 (  0.00%)        9368.00 ( 47.20%)
 SpecJBB Bops/JVM        6364.00 (  0.00%)        9368.00 ( 47.20%)
```

自动平衡 NUMA 简化了工作负载微调，可在 NUMA 计算机上实现较高的性能。如果可能，我们仍建议以静态方式微调工作负载，以便在每个节点内将其分区。但是，在所有其他情况下，自动平衡 NUMA 应该可以提升性能。

12 电源管理

电源管理旨在减少能源和散热系统的运行成本，同时使系统性能始终保持在符合当前要求的水平。因此，进行电源管理所要考虑的永远都是在系统的实际性能需求与节能方案之间取得平衡。可在系统的不同级别实施和使用电源管理。高级配置和电源接口 (ACPI) 中已经定义了设备电源管理功能及其操作系统接口的一组规范。由于服务器环境中的节能主要在处理器级别实现，因此本章将介绍主要概念，并重点说明用于分析和影响相关参数的几个工具。

12.1 CPU 级别的电源管理

在 CPU 级别，可以通过多种方式控制电源的使用。例如，通过使用空闲电源状态（C 状态）、更改 CPU 频率（P 状态）和限制 CPU（T 状态）进行控制。下列章节简要介绍了每种方法及其对节能的影响。在 https://uefi.org/sites/default/files/resources/ACPI_Spec_6_4_Jan22.pdf 上可以找到详细规范。

12.1.1 C 状态（处理器运行状态）

现代处理器有称为 `C-states` 的若干节能模式。它们反映了空闲处理器关闭不使用的组件以实现节能的能力。

当某个处理器处于 `C0` 状态时，表示它正在执行指令。在任何其他 C 状态下运行的处理器都处于空闲状态。C 后面的数字越大，CPU 休眠模式越深：将关闭更多的组件，以节省电源。深度休眠状态可以节省大量能源。缺点是会造成延迟。即，CPU 需要花费更长时间恢复到 `C0`。根据工作负载（唤醒并开始使用 CPU，随后再次回到休眠状态并持续一段短暂时间的线程）和硬件（例如网络设备的中断活动），禁用最深度休眠状态能够提升总体性能。有关操作细节，请参见第 12.3.2 节 “使用 **cpupower** 查看内核空闲统计数据”。

某些状态还包括一些具有不同节能延迟级别的子模式。支持哪些 C 状态和子模式取决于相应的处理器。但是，`C1` 始终可用。

表 12.1 “C 状态”概述了最常见的 C 状态。

表 12.1： C 状态

模式	定义
C0	运行状态。CPU 完全开启。
C1	第一种空闲状态。通过软件停止 CPU 主要内部时钟。总线接口单元和 APIC 保持全速运行。
C2	通过硬件停止 CPU 主要内部时钟。在此状态下，处理器会保留所有软件可见状态，但通过中断唤醒时可能需要更长时间。
C3	停止所有 CPU 内部时钟。处理器不需要保持其缓存的连贯性，但需要维持其他状态。某些处理器的 C3 状态存在差异，不同体现在通过中断将处理器唤醒所需的时间长短。

为了避免不必要的能耗，建议分别在启用和禁用深度休眠状态的情况下对工作负载进行测试。有关详细信息，请参见第 12.3.2 节 “使用 **cpupower** 查看内核空闲统计数据”或 **cpupower-idle-set(1)** 手册页。

12.1.2 P 状态（处理器性能状态）

当处理器运行时（处于 C0 状态），它可能处于多个 CPU 性能状态 (`P-states`) 的其中之一。C 状态反映空闲状态（除 C0 外全部处于空闲状态），而 `P-states` 则是反映与 CPU 频率和电压相关的运行状态。

P 状态数字越大，处理器的运行频率和电压越低。P 状态数字与处理器相关，处理器类型不同，其实现就会不同。但是，`P0` 始终是性能最高的状态（第 12.1.3 节 “Turbo 功能”中所述的情况除外）。P 状态数字越大，表示处理器速度越慢，能耗越低。例如，与在 `P1` 状态下运行的处理器相比，处于 `P3` 状态的处理器运行速度更慢，能耗更少。要以任何 P 状态运行，处理器必须处于 `C0` 状态，也就是说，它正在工作而不是处于空闲状态。ACPI 规范中还定义了 CPU P 状态，具体请访问 https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf。

可以单独改变 C 状态和 P 状态。

12.1.3 Turbo 功能

使用 Turbo 功能可以在其他核心处于深度休眠状态时，动态 `overtick` 活动的 CPU 核心。这可以在提高活动线程性能的同时仍符合热设计功耗 (TDP) 的限制。

但是，CPU 核心可以使用睿频的条件与特定的体系结构相关。在第 12.3 节 “cpupower 工具”中了解如何评估这些新功能的有效性。

12.2 内核中调节器

内核中调节器属于 Linux 内核 CPUfreq 基础架构，可用于在运行时动态调整处理器频率。您可将调节器视为 CPU 的某种预配置电源模式。CPUfreq 调节器使用 P 状态来更改频率和降低能耗。动态调节器可以根据 CPU 用量切换 CPU 频率，以便在不牺牲性能的情况下实现节能。

以下调节器可用于 CPUfreq 子系统：

性能调节器

将 CPU 频率静态设置为可能的最高频率，以实现最佳性能。因此，节能不是此调节器的重点。

另请参见 第 12.4.1 节 “P 状态的微调选项”。

节能调节器

将 CPU 频率静态设置为可能的最低频率。这会对性能造成严重影响，因为不管处理器有多忙，系统都不会超过此频率运行。一个重要的例外情况是，`intel_pstate` 默认采用 `powersave` 模式。这是硬件特定的决策使然，但在功能上其运行方式与 `on-demand` 调节器类似。

但是，使用此调节器通常不会产生预期的节能效果，因为通过进入 C 状态变为空闲状态时才能实现最大节能。使用节能调节器时，处理器将以最低的频率运行，因此需要更长的时间才能完成。这意味着，它会运行更长时间，直到系统能够进入空闲 C 状态为止。

微调选项：可以调整调节器可用的最小频率范围（例如，使用 **`cpupower`** 命令行工具调整）。

按需调节器

动态 CPU 频率策略的内核实现：调节器用于监控处理器的使用情况。用量超过特定阈值时，调节器会将频率设置为可用的最高频率。如果用量低于阈值，将使用下一个最低频率。如果系统还是处于利用率低的状态，则再次降低频率，直到设置最低可用频率为止。

重要：驱动程序和内核中调节器

并非所有驱动程序都会使用内核中调节器在运行时动态调整电源频率。例如 `intel_pstate` 驱动程序会自我调整电源频率。使用 **`cpupower frequency-info`** 命令可以确定系统使用的驱动程序。

12.3 cpupower 工具

`cpupower` 工具可让用户全面了解特定计算机上支持的**所有**与 CPU 功耗相关的参数，包括睿频（或加速）状态。使用该工具集可以查看和修改与内核相关的 CPUfreq 和 cpuidle 系统的设置，以及与频率调整或空闲状态无关的其他设置。集成式监控框架可以访问与内核相关的参数和硬件统计数据。因此，它非常适用于性能评测。另外，它还可帮助您识别睿频状态与空闲状态之间的依赖性。

安装 `cpupower` 软件包后，使用 **`cpupower --help`** 查看可用的 **`cpupower`** 子命令。使用 **`man cpupower`** 访问一般手册页，使用 **`man cpupower-`**`SUBCOMMAND` 访问子命令的手册页。

12.3.1 使用 cpupower 查看当前设置

`cpupower frequency-info` 命令显示内核中使用的 cpufreq 驱动程序的统计数据。此外，它还会显示 BIOS 中是否支持和启用了睿频（或加速）状态。如果不带任何选项运行，它将显示如下输出：

例 12.1：`cpupower frequency-info` 的示例输出

```
# cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 0.97 ms.
  hardware limits: 1.20 GHz - 3.80 GHz
  available cpufreq governors: performance, powersave
  current policy: frequency should be within 1.20 GHz and 3.80 GHz.
                  The governor "powersave" may decide which speed to use
```

```
                  within this range.
  current CPU frequency is 3.40 GHz (asserted by call to hardware).
  boost state support:
    Supported: yes
    Active: yes
    3500 MHz max turbo 4 active cores
    3600 MHz max turbo 3 active cores
    3600 MHz max turbo 2 active cores
    3800 MHz max turbo 1 active cores
```

要获取所有 CPU 的当前值，请使用 **cpupower -c all frequency-info**。

12.3.2 使用 cpupower 查看内核空闲统计数据

idle-info 子命令显示内核中使用的 cpuidle 驱动程序的统计。它可以在使用 cpuidle 内核框架的所有体系结构上运行。

例 12.2： cpupower idle-info 的示例输出

```
# cpupower idle-info
CPUidle driver: intel_idle
CPUidle governor: menu

Analyzing CPU 0:
Number of idle states: 6
Available idle states: POLL C1-SNB C1E-SNB C3-SNB C6-SNB C7-SNB
POLL:
Flags/Description: CPUIDLE CORE POLL IDLE
Latency: 0
Usage: 163128
Duration: 17585669
C1-SNB:
Flags/Description: MWAIT 0x00
Latency: 2
Usage: 16170005
Duration: 697658910
C1E-SNB:
Flags/Description: MWAIT 0x01
```

```
Latency: 10
Usage: 4421617
Duration: 757797385
C3-SNB:
Flags/Description: MWAIT 0x10
Latency: 80
Usage: 2135929
Duration: 735042875
C6-SNB:
Flags/Description: MWAIT 0x20
Latency: 104
Usage: 53268
Duration: 229366052
C7-SNB:
Flags/Description: MWAIT 0x30
Latency: 109
Usage: 62593595
Duration: 324631233978
```

使用 **cpupower idle-info** 找出支持的处理器空闲状态后，可以使用 **cpupower idle-set** 命令禁用单个状态。通常，用户希望禁用最深度的休眠状态，例如：

```
# cpupower idle-set -d 5
```

或者，禁用延迟等于或高于 80 的所有 CPU：

```
# cpupower idle-set -D 80
```

12.3.3 使用 cpupower 监控内核和硬件统计数据

使用 **monitor** 子命令可以报告处理器拓扑，以及监控特定时间段内的频率和空闲电源状态统计数据。默认间隔为 1 秒，但可以使用 `-i` 来更改。工具中实施了独立的处理器休眠状态和频率计数器 — 其中一些可从内核统计数据中检索，另一些可从硬件寄存器中读取。可用的监控器取决于底层硬件和系统。可以使用 **cpupower monitor -l** 列出监控器。有关各个监控器的说明，请参见 cpupower-monitor 手册页。

monitor 子命令可让您执行性能评测。要将特定工作负载的内核统计数据与硬件统计数据进行比较，请串联相应的命令，例如：

```
cpupower monitor db_test.sh
```

例 12.3： `cpupower monitor` 示例输出

```
# cpupower monitor
|Mperf                  || Idle_Stats
 ❶                        ❷
CPU | C0   | Cx   | Freq || POLL | C1   | C2   | C3
   0|  3.71| 96.29|  2833||  0.00|  0.00|  0.02| 96.32
   1| 100.0| -0.00|  2833||  0.00|  0.00|  0.00|  0.00
   2|  9.06| 90.94|  1983||  0.00|  7.69|  6.98| 76.45
   3|  7.43| 92.57|  2039||  0.00|  2.60| 12.62| 77.52
```

❶ Mperf 显示 CPU 在一段时间内的平均频率，包括加速频率。此外，它还会显示 CPU 处于活动状态 (C0) 或任意休眠状态 (Cx) 的时间百分比。由于睿频状态由 BIOS 管理，因此无法获取给定瞬间的频率值。在具备睿频功能的新式处理器上，只能通过 Mperf 监控器找出特定 CPU 的运行频率的信息。

❷ Idle_Stats 显示 cpuidle 内核子系统的统计数据。每当进入或退出空闲状态，内核都会更新这些值。因此，如果测量开始或结束时核心处于空闲状态已有一段时间，这些值可能会有点不准确。

除了上述示例中所述的（一般）监控器以外，还可以使用其他体系结构特定的监控器。有关详细信息，请参见 **`cpupower-monitor`** 手册页。

通过比较各个监控器的值，可以确定关联性和依赖关系，并评估节能机制对于特定工作负载的效果。在例 12.3 中，您可以看到，CPU `0` 处于空闲状态（`Cx` 的值接近 100%），但其运行频率却较高。这是因为，CPU `0` 和 `1` 的频率值相同，这意味着两者之间存在某种依赖关系。

12.3.4 使用 `cpupower` 修改当前设置

您可以 `root` 身份使用 **`cpupower frequency-set`** 命令来修改当前设置。此命令可让您设置供调节器选择的最小或最大 CPU 频率值，或创建新的调节器。您还可以使用 `-c` 选项指定要修改哪些处理器的设置。如此可以轻松地为所有处理器使用一致的策略，而无需单独调整每个处理器的设置。有关更多细节和可用选项，请参见手册页 `cpupower-frequency-set` 或运行 **`cpupower frequency-set`** `--help`。

12.4 特殊微调选项

下列章节重点介绍一些重要设置。

12.4.1 P 状态的微调选项

CPUfreq 子系统提供了多个针对 P 状态的微调选项：您可以在不同的调节器之间切换、影响要使用的最小或最大 CPU 频率，或单独更改某个调节器参数。

要在运行时切换到另一个调节器，请结合 `-g` 选项使用 **`cpupower frequency-set`**。例如，运行以下命令（以 `root` 身份）会激活性能调节器：

```
# cpupower frequency-set -g performance
```

要设置可供调节器选择的最小或最大 CPU 频率值，请分别使用 `-d` 或 `-u` 选项。

12.5 查错

是否已启用 BIOS 选项?

要使用 C 状态或 P 状态，请检查 BIOS 选项：

- 要使用 C 状态，请确保启用 `CPU C State` 或类似选项，以便在空闲时实现节能。
- 要使用 P 状态和 CPUfreq 调节器，请确保启用 `Processor Performance States` 选项或类似选项。
- 即使 P 状态和 C 状态可用，也可能存在 CPU 频率由平台固件管理的情况，从而导致性能欠佳。例如，如果已加载 `pcc-cpufreq`，则操作系统只会向固件发出提示，而后者可以随意忽略提示。要解决此问题，可以针对 BIOS 中管理的 CPU 频率，选择“操作系统管理”或类似选项。重引导后，将使用替代的驱动程序，但仍应仔细测量性能影响。

升级 CPU 时，请务必同时升级 BIOS。BIOS 需要知道新 CPU 及其频率步进值，以便将此信息传递给操作系统。

是否有日志文件信息?

在 systemd 日记（请参见《管理指南》, 第 21 章 “**journalctl**：查询 systemd 日记”）中检查有关 CPUfreq 子系统的任何输出。其中只会报告严重错误。

如果您怀疑计算机上的 CPUfreq 子系统有问题，还可以启用额外的调试输出。为此，请使用 **cpufreq.debug=7** 作为引导参数，或以 root 身份执行以下命令：

```
# echo 7 > /sys/module/cpufreq/parameters/debug
```

这会导致 CPUfreq 在状态转换时将更多信息记录到 **dmesg**，这些信息有助于进行诊断。但这些附加的内核消息输出可能相当详尽，因此请仅当您确信存在问题时才采用这种做法。

12.6 更多信息

配备基板管理控制器 (BMC) 的平台可能会提供可通过服务处理器访问的其他电源管理配置选项。这些配置与特定的供应商相关，因此不属于本指南讨论的主题。有关详细信息，请参见供应商提供的手册。

有关 powerTOP 的详细信息，请访问 https://01.org/powertop。

12.7 使用 powerTOP 监控能耗

powerTOP 可帮助识别能耗不必要升高的原因。在笔记本电脑上，此工具特别有用，因为尽量减少能耗对笔记本电脑而言更为重要。此工具支持 Intel 和 AMD 处理器。以一般方式安装此工具：

```
> sudo zypper in powertop
```

powerTOP 结合了多种信息来源（程序分析、设备驱动程序、内核选项、将处理器从休眠状态唤醒的中断数目和来源），并提供多种方式用于查看这些信息。您可以在交互模式下启动此工具，它将在 ncurses 会话中运行（请参见图 12.1 “处于交互模式的 powerTOP”）：

```
> sudo powertop
```

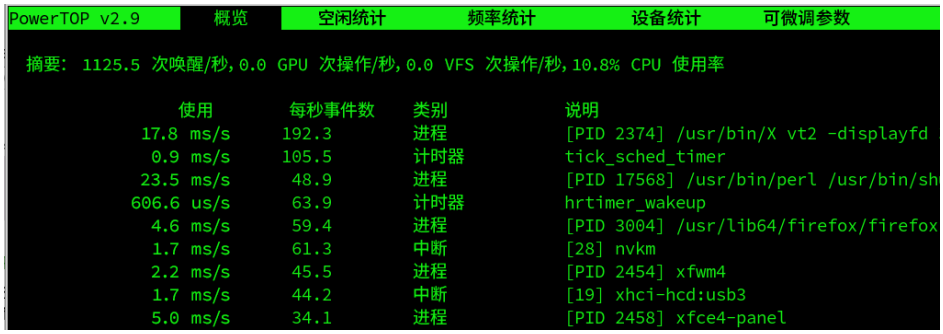

图 12.1： 处于交互模式的 POWERTOP

powerTOP 支持将报告导出为 HTML 和 CSV 格式。以下示例生成了一个运行 240 秒的报告：

```
> sudo powertop --iteration=1 --time=240 --html=POWERREPORT.HTML
```

对于在一段时间内运行独立报告此工具会很有用。以下示例会运行 powerTOP 10 次，每次运行 20 秒，并且每次运行时都会创建一份相应的 HTML 报告：

```
> sudo powertop --iteration=10 --time=20 --html=POWERREPORT.HTML
```

这会创建 10 个带时戳的报告：

```
powerreport-20200108-104512.html
powerreport-20200108-104451.html
powerreport-20200108-104431.html
[...]
```

HTML 报告如图 12.2 “HTML powerTOP 报告”中所示：

图 12.2： HTML POWERTOP 报告

HTML 报告的“微调”选项卡以及交互模式下的“可调参数”选项卡都提供了用于测试各种电源设置的命令。HTML 报告会列显命令，您可将其复制到 root 命令行进行测试，例如 **`echo '0' > '/proc/sys/kernel/nmi_watchdog'`**。ncurses 模式提供了一种在 Good 和 Bad 之间进行切换的简单方式。Good 会运行命令以启用省电功能，而 Bad 则会关闭省电功能。通过一条命令启用所有 powerTOP 设置：

```
> sudo powertop --auto-tune
```

重引导后，所有这些更改都不会留存。要进行永久性更改，请使用 **`sysctl`**、**`udev`** 或 `systemd` 在系统引导时运行选定的命令。powerTOP 包含一个 `systemd` 服务文件：`/usr/lib/systemd/system/powertop.service`。以下命令使用 **`--auto-tune`** 选项启动 powerTOP：

```
ExecStart=/usr/sbin/powertop --auto-tune
```

在启动 `systemd` 服务之前，请仔细测试，以确定它是否能够获得所需的结果。不应使用 USB 键盘，鼠标也不应进入节能模式，以免不断唤醒它们并干扰其他设备。为了更方便地进行测试和编辑配置，请使用 **`awk`** 提取 HTML 报告中的命令：

```
> awk -F '</?td ?>' '/tune/ { print $4 }' POWERREPORT.HTML
```

在校准模式下，powerTOP 会设置多个运行作业，即针对背光、CPU、Wi-Fi、USB 设备和磁盘使用不同的空闲设置，此外，在电池供电时，powerTOP 还可帮助您确定最佳的亮度设置：

```
> sudo powertop --calibrate
```

您可以调用文件来创建工作负载，以提高校验的准确性：

```
> sudo powertop --calibrate --workload=FILENAME --html=POWERREPORT.HTML
```

有关更多信息，请参见：

- powerTOP 项目页面：https://01.org/powertop
- 第 2.6.2 节 “系统控制参数：/proc/sys/”
- 《管理指南》, 第 19 章 “systemd 守护程序”
- 《管理指南》, 第 29 章 “ 使用 udev 进行动态内核设备管理”

V 内核微调

13 微调 I/O 性能 136

14 微调任务调度程序 141

15 微调内存管理子系统 153

16 微调网络 164

13 微调 I/O 性能

I/O 调度用于控制将输入/输出操作提交到存储空间的方式。SUSE Linux Enterprise Desktop 提供了多种适合不同工作负载的 I/O 算法（称为 `elevators`）。电梯算法有助于减少搜寻操作并可指定 I/O 请求的优先级。

如何选择最合适的 I/O 电梯算法不仅取决于工作负载，还取决于硬件。例如，每个 ATA 磁盘系统、SSD、RAID 阵列或网络存储系统都需要不同的微调策略。

13.1 切换 I/O 调度

SUSE Linux Enterprise Desktop 在引导时会选择使用默认的 I/O 调度程序，您可以针对每个块设备即时更改此设置。例如，可为托管系统分区的设备以及托管数据库的设备设置不同的算法。

根据设备是否报告为旋转磁盘为每个设备选择默认 I/O 调度程序。如果是旋转磁盘，则选取 `BFQ` I/O 调度程序。其他设备默认选择 `MQ-DEADLINE` 或 `NONE`。

要更改正在运行的系统中特定设备的电梯算法，请运行以下命令：

```
> sudo echo SCHEDULER > /sys/block/DEVICE/queue/scheduler
```

其中 `SCHEDULER` 是 `bfq`、`none`、`kyber` 或 `mq-deadline` 之一，`DEVICE` 是块设备（例如 `sda`）。此项更改在重引导期间不会保留。要对特定设备进行永久的 I/O 调度程序更改，请将 `/usr/lib/udev/rules.d/60-io-scheduler.rules` 复制到 `/etc/udev/rules.d/60-io-scheduler.rules`，然后根据需要编辑后一个文件。

13.2 使用 blk-mq I/O 路径的可用 I/O 电梯算法

下面是 SUSE Linux Enterprise Desktop 上适用于使用 blk-mq I/O 路径的设备的电梯算法列表。如果某个电梯算法包含可调参数，可使用以下命令设置这些参数：

```
echo VALUE > /sys/block/DEVICE/queue/iosched/TUNABLE
```

在上述命令中，`VALUE` 是 `TUNABLE` 的所需值，`DEVICE` 是块设备。

要确定某个设备（例如 sda）适用哪些电梯算法，请运行以下命令（当前选择的调度程序列在方括号中）：

```
> cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber bfq none
```

13.2.1 MQ-DEADLINE

MQ-DEADLINE 是一个面向延迟优化的 I/O 调度程序。MQ-DEADLINE 具有以下可调参数：

表 13.1：MQ-DEADLINE 可调参数

文件	说明
writes_starved	控制优先处理读取操作而不是写入操作的次数。值 3 表示在执行写入操作之前可以执行三次读取操作，读取操作按照相同的选择准则进行调度。 默认值为 3。
read_expire	设置读取操作的最后期限（当前时间加 read_expire 值），以毫秒为单位。 默认值为 500。
write_expire	设置写入操作的最后期限（当前时间加 write_expire 值），以毫秒为单位。 默认值为 5000。
front_merges	启用 (1) 或禁用 (0) 前端合并请求尝试。 默认值为 1。
fifo_batch	设置每批的最大请求数（仅检查批的最后期限失效时间）。使用此参数可在延迟与吞吐量之间实现平衡。如果设置为 1（即每批一个请求），系统将采用“先到先得”模式，延迟将降到最低；设置更大的值会提高吞吐量。 默认值为 16。

13.2.2 NONE

如果选择 NONE 作为 blk-mq 的 I/O 电梯算法选项，则不会使用任何 I/O 调度程序，并且 I/O 请求将传递给设备，而不做进一步的 I/O 调度交互。

NONE 是 NVM Express 设备的默认值。与其他 I/O 电梯算法选项相比，此选项不会产生开销，被认为是通过多个队列向此类设备传递 I/O 请求最快的方式。

NONE 没有可调参数。

13.2.3 BFQ（预算公平队列）

BFQ 是以公平性为导向的调度程序。其描述为“基于 CFQ 片服务模式的按比例存储 I/O 调度算法。但 BFQ 会向进程分配预算（以扇区数计）而不是时间片。”（来源：linux-4.12/block/bfq-iosched.c (https://github.com/torvalds/linux/blob/6f7da290413ba713f0cdd9ff1a2a9bb129ef4f6c/block/bfq-iosched.c#L31)）

BFQ 允许向调度决策期间要考虑的任务分配 I/O 优先级（请参见第 9.3.3 节 “使用 **ionice** 指定磁盘访问优先级”）。

BFQ 调度程序具有以下可调参数：

表 13.2： BFQ 可调参数

文件	说明
slice_idle	值以毫秒为单位，指定处于空闲状态多长时间，以等待对空队列发出下一个请求。 默认值为 8。
slice_idle_us	与 slice_idle 相同，但以微秒为单位。 默认值为 8000。
low_latency	启用 (1) 或禁用 (0) BFQ 的低延迟模式。此模式优先处理特定的应用程序（例如，如果是交互式的），使其保持较低的延迟。 默认值为 1。
back_seek_max	向后搜寻的最大值（以 KB 为单位）。

文件	说明
	默认值为 16384。
back_seek_penalty	用于计算向后搜寻的成本。 默认值为 2。
fifo_expire_async	用于设置异步请求超时的值（以毫秒为单位）。 默认值为 250。
fifo_expire_sync	值以毫秒为单位，指定同步请求的超时。 默认值为 125。
timeout_sync	选择某个任务（队列）后，为其提供服务的最长时间（以毫秒为单位）。 默认值为 124。
max_budget	在 timeout_sync 内为其提供服务的扇区数上限。如果设置为 0，BFQ 会根据 timeout_sync 和预估的峰值速率自动计算一个值。 默认值为 0（设置为自动微调）。
strict_guarantees	启用 (1) 或禁用 (0)，需要 BFQ 特定队列处理，才能在特定条件下做出更严格的带宽共享保证。 默认值为 0。

13.2.4 KYBER

KYBER 是以延迟为导向的 I/O 调度程序。使用它可以设置读取操作和同步写入操作的目标延迟，并限制 I/O 请求以尝试符合这些目标延迟。

表 13.3： KYBER 可调参数

文件	说明
`read_lat_nsec`	设置读取操作的目标延迟，以纳秒为单位。 默认值为 `2000000`。
`write_lat_nsec`	设置写入操作的目标延迟，以纳秒为单位。 默认值为 `10000000`。

13.3 I/O 屏障微调

在完成 fsync 之后或在事务提交期间，某些文件系统（例如 Ext3 或 Ext4）会向磁盘发送写屏障。写屏障会强制按正确排序写入，以确保能安全使用非永久磁盘写缓存（但会导致性能下降）。如果您的磁盘以这样或那样的方式采用了电池供电机制，禁用屏障可以安全地改善性能。

重要：XFS 中的 nobarrier 已弃用

XFS 的 `nobarrier` 选项已弃用，它在 SUSE Linux Enterprise 15 SP2 和更高版本中不是有效的挂载选项。任何明确指定该标志的 XFS 挂载命令都可能无法挂载文件系统。为防止发生这种问题，请确保不存在任何包含 `nobarrier` 选项的脚本或 fstab 项。

您可以使用 `nobarrier` 挂载选项禁用发送写屏障。

警告：禁用屏障可能导致数据丢失

当磁盘无法保证在电源故障情况下正确写入缓存时，禁用屏障可能会导致严重的文件系统损坏和数据丢失。

14 微调任务调度程序

新式操作系统（例如 SUSE® Linux Enterprise Desktop）往往会同时运行许多任务。例如，您可以在搜索文本文件的同时接收电子邮件以及将大型文件复制到外部硬盘。这些简单任务要求系统运行许多额外的进程。为了给每个任务提供所需的系统资源，Linux 内核需要使用一种工具向各个任务分发可用系统资源。这恰恰是**任务调度程序**的职责所在。

下列章节解释了与进程调度相关的最重要术语。其中还介绍了有关任务调度程序策略和调度算法的信息、SUSE Linux Enterprise Desktop 使用的任务调度程序，并提供了其他来源的相关信息的参考。

14.1 简介

Linux 内核用于控制在系统上管理任务（或进程）的方式。任务调度程序（有时称作**进程调度程序**）是决定接下来要运行哪个任务的内核组件。它负责以最优方式使用系统资源，以保证能够同时执行多个任务。因此，它是任何多任务操作系统的核心组件。

14.1.1 抢占

任务调度背后的理论很简单。如果系统中存在多个可运行的进程，必须始终至少有一个进程正在运行。如果可运行进程的数目超过了系统中处理器的数目，则并非所有进程都能始终运行。

因此，某些进程需要暂时停止（或**挂起**），使其他进程能够再次运行。由调度程序决定下一次运行队列中的哪个进程。

如前所述，Linux 与其他所有类 Unix 操作系统一样，是一种**多任务**操作系统。即，多个任务可以同时运行。Linux 提供所谓的**抢占式**多任务，让调度程序决定何时挂起某个进程。这种强制挂起称为**抢占**。从一开始就为所有 Unix 风格提供了抢占式多任务。

14.1.2 时间片

进程在被**抢占**前的运行时长是预先设定的，这一时长称为进程的**时间片**，代表分配给每个进程的处理器时间。通过分配时间片，调度程序可以针对运行中的系统做出全局决策，同时防止个别进程独占处理器资源。

14.1.3 进程优先级

调度程序基于进程的优先级评估进程。任务调度程序使用复杂算法计算进程的当前优先级。因此，每个进程都会被赋予一个值，据此决定其是否被“允许”在处理器上运行。

14.2 进程分类

进程按用途和行为分类。尽管界线并不总是清晰明确，但一般会采用两条准则进行分类。这两条准则彼此独立，但并非互不包容。

一种方法是将进程分类为 **I/O 密集型**或**处理器密集型**。

I/O 密集型

I/O 代表输入/输出设备，例如键盘、鼠标或者光盘和硬盘。**I/O 密集型进程**将大部分时间花在提交和等待请求上。它们的运行频率较高，但时间间隔较短，不会阻塞等待 I/O 请求的其他进程。

处理器密集型

处理器密集型任务将时间用在执行代码上，运行到被调度程序抢占为止。它们不会阻塞等待 I/O 请求的进程，因此其运行频率可能更低，但间隔的时间更长。

另一种方法是按类型将进程划分为**交互式**、**批处理**和**实时**进程。

- **交互式**进程将大量时间花在等待 I/O 请求（例如键盘或鼠标操作）上。调度程序必须按用户请求快速唤醒此类进程，否则用户会认为环境无响应。典型延迟大约为 100 毫秒。办公应用程序、文本编辑器或图像处理程序都是典型的交互式进程。
- **批处理**进程通常在后台运行，不需要做出响应。调度程序会为它们分配较低的优先级。多媒体转换器、数据库搜索引擎或日志文件分析器都是批处理进程的典型示例。
- **实时**进程绝不应该被低优先级进程阻塞，调度程序需保证这些进程在很短时间获得响应。用于编辑多媒体内容的应用程序就是实时进程的典型示例。

14.3 完全公平调度程序

从 Linux 内核版本 2.6.23 开始，采用了一种新方法来调度可运行的进程。完全公平调度程序 (CFS) 成为了默认 Linux 内核调度程序。自此以后发生了重大变化和改进。本章中的信息适用于采用 2.6.32 和更高内核版本（包括 3.x 内核）的 SUSE Linux Enterprise Desktop。调度程序环境分为多个部分，引入了三项主要新功能：

模块化调度程序核心

调度程序的核心已通过**调度类**得到增强。这些类是模块化的，代表调度策略。

完全公平调度程序

在内核 2.6.23 中引入并在 2.6.24 中进行了扩展的 CFS 会尝试确保每个进程获得其“公平”份额的处理器时间。

组安排

例如，如果您根据运行进程的用户将进程分组，则 CFS 会尝试为其中每个组提供等量的处理器时间。

因此，CFS 能够针对服务器和桌面优化调度。

14.3.1 CFS 的工作原理

CFS 尝试为每个可运行的任务保证公平性。为了找出最平衡的任务调度方式，它使用了**红黑树**概念。红黑树是一种自我平衡式数据搜索树，能够以合理方式提供插入项和去除项，使其自身保持适当的平衡。

CFS 在调度任务时会累加“虚拟运行时” (**vruntime**)。选择运行的下一个任务始终是迄今为止累加 vruntime 最小的那个任务。在将任务插入**运行队列**（后续要执行的进程的规划时间线）时，通过平衡红黑树可保证 vruntime 最小的任务始终是红黑树中的第一项。

任务的累计 vruntime 与该任务的优先级相关。高优先级任务的 vruntime 的累加速度比低优先级任务要慢，导致更频繁地选择在处理器上运行高优先级任务。

14.3.2 将进程分组

从 Linux 内核版本 2.6.24 开始，可对 CFS 进行微调，以保证为组而不仅仅是任务提供公平性。为此，将可运行的任务分组以构成实体，而 CFS 会尝试为这些实体而不是各个可运行的任务提供公平性。调度程序还会尝试为这些实体中的各个任务提供公平性。

内核调度程序可让您使用控制组将可运行的任务分组。有关详细信息，请参见第 10 章 “内核控制组”。

14.3.3 内核配置选项

您可以通过内核配置选项来设置任务调度程序行为的基本方面。设置这些选项属于内核编译过程的一部分。由于内核编译过程是一个复杂任务，超出了本文档的讨论范畴，因此请参考相关的信息来源。

警告：内核编译

如果您在并非 SUSE Linux Enterprise Desktop 随附的内核上运行该操作系统（例如，在自我编译的内核上运行），您会失去全部支持权利。

14.3.4 术语

有关任务调度策略的文档经常会使用一些技术术语，您需要熟悉这些术语才能正确理解文档中的信息。下面是其中的一些术语：

延迟

从预定运行进程到实际执行该进程延迟的时间。

粒度

可通过以下公式表达粒度与延迟之间的关系：

```
gran = ( lat / rtasks ) - ( lat / rtasks / rtasks )
```

其中 **gran** 表示粒度，**lat** 表示延迟，**rtasks** 是正在运行的任务数。

14.3.4.1 调度策略

Linux 内核支持以下调度策略：

SCHED_FIFO

适用于特殊时间关键型应用程序的调度策略。它使用“先进先出”调度算法。

SCHED_BATCH

适用于 CPU 密集型任务的调度策略。

SCHED_IDLE

适用于**极**低优先级任务的调度策略。

SCHED_OTHER

大多数进程使用的默认 Linux 分时调度策略。

SCHED_RR

与 SCHED_FIFO 类似，但使用循环复用调度算法。

14.3.5 使用 chrt 更改进程的实时属性

chrt 命令设置或检索运行中进程的实时调度属性，或者使用指定的属性运行某个命令。您可以获取或检索进程的调度策略和优先级。

在以下示例中，使用了 PID 为 16244 的进程。

要**检索**现有任务的实时属性，请执行以下命令：

```
# chrt -p 16244
pid 16244's current scheduling policy: SCHED_OTHER
pid 16244's current scheduling priority: 0
```

在对进程设置新的调度策略之前，需要找出每个调度算法的最小和最大有效优先级：

```
# chrt -m
SCHED_SCHED_OTHER min/max priority : 0/0
SCHED_SCHED_FIFO min/max priority : 1/99
SCHED_SCHED_RR min/max priority : 1/99
SCHED_SCHED_BATCH min/max priority : 0/0
SCHED_SCHED_IDLE min/max priority : 0/0
```

在以上示例中，SCHED_OTHER、SCHED_BATCH、SCHED_IDLE 策略仅允许优先级为 0，而 SCHED_FIFO 和 SCHED_RR 的优先级可为 1 到 99。

要设置 SCHED_BATCH 调度策略，请执行以下命令：

```
# chrt -b -p 0 16244
pid 16244's current scheduling policy: SCHED_BATCH
pid 16244's current scheduling priority: 0
```

有关 **chrt** 的详细信息，请参见其手册页 (**man 1 chrt**)。

14.3.6 使用 sysctl 进行运行时微调

用于在运行时检查和更改内核参数的 **sysctl** 接口引入了重要的变量，您可以通过这些变量更改任务调度程序的默认行为。**sysctl** 的语法非常简单。必须以 root 身份在命令行上输入所有以下命令。

要从内核变量读取值，请输入

```
# sysctl VARIABLE
```

要赋值，请输入

```
# sysctl VARIABLE=VALUE
```

要获取与调度程序相关的所有变量的列表，请运行 **sysctl** 命令，并使用 **grep** 过滤输出：

```
# sysctl -A | grep "sched" | grep -v "domain"
kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_compat_yield = 0
kernel.sched_latency_ns = 24000000
kernel.sched_migration_cost_ns = 500000
kernel.sched_min_granularity_ns = 8000000
kernel.sched_nr_migrate = 32
kernel.sched_rr_timeslice_ms = 25
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_schedstats = 0
kernel.sched_shares_window_ns = 10000000
```

```
kernel.sched_time_avg_ms = 1000
kernel.sched_tunable_scaling = 1
kernel.sched_wakeup_granularity_ns = 10000000
```

以 “_ns” 和 “_us” 结尾的变量分别接受以纳秒和微秒为单位的值。

下面提供了最重要任务调度程序 **sysctl** 微调变量的列表（位于 /proc/sys/kernel/ 中）和简短说明：

sched_cfs_bandwidth_slice_us

使用 CFS 带宽控制时，此参数会控制从任务的控制组带宽池传送到运行队列的运行时间长短（带宽）。指定较小值可以在任务之间精细共享全局带宽，指定较大值可减少传送开销。请访问 https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt 。

sched_child_runs_first

新派生的子项在父项继续执行之前运行。将此参数设置为 1 有利于其中的子项在派生后完成执行的应用程序。

sched_compat_yield

通过将主动让出 CPU 的任务移到可运行队列的末尾（红黑树中的最右边），实现旧 **O(1)** 调度程序那种激进的 CPU 让出行为。出现严重的资源争用（例如锁定）时，通过给予其他进程运行机会，依赖 sched_yield(2) syscall 行为的应用程序可能会获得性能提高。假设这种调用是在环境切换时发生的，滥用调用可能会给工作负载造成损害。请仅当您发现性能下降时，才使用此变量。默认值为 0。

sched_migration_cost_ns

在迁移决策中，任务自上次执行后仍被视为“缓存热点”的时长。“热点”任务被迁移到其他 CPU 的可能性较低，因此增大该变量会减少任务迁移。默认值为 500000（纳秒）。如果存在可运行进程时 CPU 空闲时间超过预期，请尝试减小此值。如果任务过于频繁地在 CPU 或节点之间弹跳，请尝试增大此值。

sched_latency_ns

CPU 密集型任务的目标抢占延迟。增大此变量会增加 CPU 密集型任务的时间片。任务的时间片是其调度时段的加权公平份额：

时间片 = 调度时段 * (任务的权重/任务在运行队列中的总权重)

任务的权重取决于任务的 nice 级别和调度策略。SCHED_OTHER 任务的最小任务权重为 15，对应于 nice 19。最大任务权重为 88761，对应于 nice -20。

时间片随着负载增加而变小。当可运行任务的数目超过 `sched_latency_ns`/`sched_min_granularity_ns` 时，切片将变为 number_of_running_tasks * `sched_min_granularity_ns`。在此之前，切片等于 `sched_latency_ns`。

此值还指定在权利计算中将某个休眠任务视为正在运行的最长时间量。增大此变量会增加抢占某个唤醒任务之前该任务可能消耗的时间，因此会增加 CPU 密集型任务的调度程序延迟。默认值为 `6000000`（纳秒）。

`sched_min_granularity_ns`

CPU 密集型任务的最小抢占粒度。有关详细信息，请参见`sched_latency_ns`。默认值为 `4000000`（纳秒）。

`sched_wakeup_granularity_ns`

唤醒抢占粒度。增大此变量会减小唤醒抢占，从而减轻计算密集型任务的干扰。减小此变量可改善延迟关键型任务的唤醒延迟和吞吐量，尤其是当短工作周期负载组件必须与 CPU 密集型组件竞争时。默认值为 `2500000`（纳秒）。

警告：设置适当的唤醒粒度值

如果设置值超过 `sched_latency_ns` 的一半，将导致不会发生唤醒抢占。短工作周期任务无法有效与 CPU 资源霸占者竞争。

`sched_rr_timeslice_ms`

在 SCHED_RR 任务被抢占并置于任务列表末尾之前可运行的量子。

`sched_rt_period_us`

测量实时任务带宽强制的时间段。默认值为 `1000000`（微秒）。

`sched_rt_runtime_us`

在 sched_rt_period_us 时间段分配给实时任务的量子。设置为 -1 会禁用 RT 带宽强制。默认情况下，RT 任务每秒可能消耗 95%CPU，因而剩下 5%CPU/秒（或 0.05 秒）供 SCHED_OTHER 任务使用。默认值为 `950000`（微秒）。

`sched_nr_migrate`

控制为实现负载平衡可跨处理器迁移的任务数。由于平衡机制在禁用中断 (softirq) 的情况下迭代运行队列，它可能造成实时任务的 IRQ 延迟。因此，增大此值可能会提升大型 SCHED_OTHER 线程的性能，代价是会增加实时任务的 IRQ 延迟。默认值为 32。

`sched_time_avg_ms`

此参数用于设置计算实时任务运行时间的平均值所依据的时间段。该平均值可帮助 CFS 做出负载平衡决策，并指示 CPU 处理高优先级实时任务的繁忙程度。

此参数的最佳设置在很大程度上与工作负载相关，并取决于实时任务的运行频率和运行时长等因素。

警告：一些调度程序参数已移至 debugfs

如果操作系统的默认 Linux 内核为 5.13 或更高版本（可以使用命令 `rpm -q kernel-default` 查看），您可能会看到内核日志中类似于以下示例的消息：

```
[ 20.485624] The sched.sched_min_granularity_ns sysctl was moved to
 debugfs in kernel 5.13 for CPU scheduler debugging only. This sysctl will
 be removed in a future SLE release.
[ 20.485632] The sched.sched_wakeup_granularity_ns sysctl was moved to
 debugfs in kernel 5.13 for CPU scheduler debugging only. This sysctl will
 be removed in a future SLE release.
```

之所以发生此情况，是因为六个调度程序参数已从 `/proc/sys/kernel/sched_*` 移至 `/sys/kernel/debug/sched/*`。受影响的调度程序参数如下：

- `sched_latency_ns`
- `sched_migration_cost_ns`
- `sched_min_granularity_ns`
- `sched_nr_migrate`
- `sched_tunable_scaling`
- `sched_wakeup_granularity_n`

为了暂时方便起见，SUSE Linux Enterprise Desktop 中仍然提供这些调度程序参数的 `sysctl`。但是，鉴于 CPU 调度程序实现中计划的更改，无法保证 SUSE Linux Enterprise Desktop 的未来版本中会提供 `sysctl` 或 `debugfs` 选项。

如果您当前有任何系统微调配置依赖于这六个调度程序参数，我们强烈建议您寻找替代方法来实现目标，并停止依赖这些参数来处理生产工作负载。

14.3.7 调试界面和调度程序统计数据

CFS 随附了新的增强型调试界面，并提供运行时统计数据。相关文件已添加到 `/proc` 文件系统，使用 **`cat`** 或 **`less`** 命令即可检查这些文件。下面提供了相关 `/proc` 文件的列表及其简短说明：

`/proc/sched_debug`

包含影响任务调度程序行为的所有可调变量（请参见第 14.3.6 节 “使用 **`sysctl`** 进行运行时微调”）的当前值、CFS 统计数据，以及有关所有可用处理器上的运行队列的信息（CFS、RT 和最后期限）。此外还会显示每个处理器上所运行任务的摘要、任务名称和 PID，以及与调度程序相关的统计数据。最先显示的是 `tree-key` 列，其中指出了任务的虚拟运行时及其在内核中的名称，在红黑树中可按此键对所有可运行任务进行排序。`switches` 列指出切换总次数（无论强制还是自愿），`prio` 表示进程优先级。`wait-time` 值表示任务等待调度的时间。最后，`sum-exec` 和 `sum-sleep` 分别统计了任务在处理器上运行或者处于休眠状态的总时间（以纳秒为单位）。

```
# cat /proc/sched_debug
Sched Debug Version: v0.11, 6.4.0-150700.38-default #1
ktime                                   : 23533900.395978
sched_clk                               : 23543587.726648
cpu_clk                                 : 23533900.396165
jiffies                                 : 4300775771
sched_clock_stable                      : 0

sysctl_sched
  .sysctl_sched_latency                    : 6.000000
  .sysctl_sched_min_granularity            : 2.000000
  .sysctl_sched_wakeup_granularity         : 2.500000
  .sysctl_sched_child_runs_first           : 0
  .sysctl_sched_features                   : 154871
  .sysctl_sched_tunable_scaling            : 1 (logarithmic)

cpu#0, 2666.762 MHz
  .nr_running                    : 1
  .load                          : 1024
  .nr_switches                   : 1918946
[...]
```

```
cfs_rq[0]:/
  .exec_clock                    : 170176.383770
  .MIN_vruntime                  : 0.000001
  .min_vruntime                  : 347375.854324
  .max_vruntime                  : 0.000001
[...]

rt_rq[0]:/
  .rt_nr_running                 : 0
  .rt_throttled                  : 0
  .rt_time                       : 0.000000
  .rt_runtime                    : 950.000000

dl_rq[0]:
  .dl_nr_running                 : 0

  task   PID         tree-key  switches  prio     wait-time        [...]
---------------------------------------------------------------------------
R  cc1 63477     98876.717832       197   120      0.000000         ...
```

`/proc/schedstat`

显示与当前运行队列相关的统计数据。另外还显示所有已连接处理器的 SMP 系统域特定统计数据。由于输出格式不太直观，请阅读 `/usr/src/linux/Documentation/scheduler/sched-stats.txt` 的内容来了解详细信息。

`/proc/PID/sched`

显示有关进程的调度信息及其 ID (`PID`)。

```
# cat /proc/$(pidof gdm)/sched
gdm (744, #threads: 3)
-------------------------------------------------------------------
se.exec_start                                :          8888.758381
se.vruntime                                  :          6062.853815
se.sum_exec_runtime                          :             7.836043
se.statistics.wait_start                     :             0.000000
se.statistics.sleep_start                    :          8888.758381
se.statistics.block_start                    :             0.000000
se.statistics.sleep_max                      :          1965.987638
```

```
[...]
se.avg.decay_count                              :                8477
policy                                          :                   0
prio                                            :                 120
clock-delta                                     :                 128
mm->numa_scan_seq                               :                   0
numa_migrations, 0
numa_faults_memory, 0, 0, 1, 0, -1
numa_faults_memory, 1, 0, 0, 0, -1
```

14.4 更多信息

要获得有关 Linux 内核任务调度的简要介绍，需浏览多个信息来源。下面是其中一些资源：

- 有关任务调度程序系统调用的介绍，请参见相关手册页（例如 **`man 2 sched_setaffinity`**）。
- 有关 Linux 调度程序策略和算法的实用讲座，请访问 https://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture08.pdf。
- Robert Love 所著的 Linux Kernel Development (ISBN-10: 0-672-32512-8) 对 Linux 进程调度进行了精辟概述。请访问 https://www.informit.com/articles/article.aspx?p=101760。
- 由 Daniel P. Bovet 和 Marco Cesati 撰写的 Understanding the Linux Kernel (ISBN 978-0-596-00565-8) 中全面概述了 Linux 内核的内部结构。
- `/usr/src/linux/Documentation/scheduler` 下的文件介绍了有关任务调度程序的技术信息。

15 微调内存管理子系统

要了解和微调内核的内存管理行为，请务必先大致了解其工作原理及与其他子系统的协作方式。

内存管理子系统也称为虚拟内存管理器，下文称为 “VM”。VM 的作用是管理整个内核以及用户程序的物理内存 (RAM) 分配。它还负责为用户进程提供虚拟内存环境（使用 Linux 扩展通过 POSIX API 进行管理）。最后，当出现内存不足情况时，VM 会通过清理缓存或换出“匿名”内存来释放 RAM。

检查和微调 VM 时要了解的最重要的事项是如何管理 VM 的缓存。VM 缓存的基本目标是最大程度地减少交换和文件系统操作（包括网络文件系统）产生的 I/O 开销。可通过避免 I/O 或以更好的模式提交 I/O 来实现此目标。

这些缓存会根据需要使用并填满可用内存。缓存和匿名内存可用的内存越多，缓存和交换的运行效率就越高。但是，如果遇到内存不足的情况，就会清理缓存或换出内存。

对于特定的工作负载而言，如果想要改善性能，可以采取的第一项措施就是增加内存并降低必须清理或交换内存的频率。第二项措施是通过更改内核参数来改变缓存的管理方式。

最后，还应检查并微调工作负载本身。如果允许应用程序运行更多进程或线程，且每个进程在其各自的文件系统区域中运行，则 VM 缓存的效率可能会下降。内存开销也随之升高。如果应用程序可为自身分配缓冲区或缓存，则缓存越大意味着 VM 缓存的可用内存越少。但是，更多的进程和线程可能意味着有更大的机会以重叠和管道形式执行 I/O，因此可以更好地利用多个核心。需要进行试验才能获得最佳效果。

15.1 内存用量

内存分配可分为以下几类：“固定的”（也称为“不可回收的”）、“可回收的”或“可交换的”。

15.1.1 匿名内存

匿名内存往往是程序堆和堆栈内存（例如 `>malloc()`）。匿名内存是可回收的，诸如 `mlock` 或没有可用的交换空间等特殊情况除外。匿名内存必须先写入交换区，然后才能回收。由于分配和访问模式的原因，交换 I/O（包括换入和换出页）的效率往往比页缓存 I/O 的效率低。

15.1.2 页缓存

文件数据的缓存。从磁盘或网络读取文件时，内容将存储在页缓存中。如果页缓存中的内容为最新数据，则无需访问磁盘或网络。tmpfs 和共享内存段将计入页缓存中。

写入文件时，新数据会先存储在页缓存中，然后再写回到磁盘或网络（使其成为写回缓存）。当某一页包含尚未写回的新数据时，该页称为“脏页”。不属于脏页的其他页为“干净页”。当出现内存不足时，可以回收干净的页缓存页，只需将其释放即可。脏页必须先变为干净页，然后才能回收。

15.1.3 缓冲区缓存

这是适用于块设备（例如 /dev/sda）的一种页缓存。文件系统在访问其磁盘上的元数据结构（例如 inode 表、分配位图等）时，通常会使用缓冲区缓存。可以像回收页缓存一样回收缓冲区缓存。

15.1.4 缓冲区头

缓冲区头是一种小型辅助结构，往往是在访问页缓存时分配的。如果页缓存页或缓冲区缓存页是干净的，则通常可以轻松回收。

15.1.5 写回

当应用程序在文件中写入数据时，页缓存会变为脏缓存，而缓冲区缓存可能也会变为脏缓存。当脏内存量达到指定的页数量（以字节为单位）(**vm.dirty_background_bytes**)、或者当脏内存量与总内存之比达到特定比率 (**vm.dirty_background_ratio**)，或者当页处于脏状态的时间超过指定的值 (**vm.dirty_expire_centisecs**) 时，内核将从其页最先变为脏页的文件开始完成页写回。后台字节数和比率是互斥的，设置其中一个会重写另一个。刷新程序线程在后台执行写回，允许应用程序继续运行。如果 I/O 跟不上应用程序将页缓存变脏的进度，并且脏数据达到关键设置（**vm.dirty_bytes** 或 **vm.dirty_ratio**），将会开始限制应用程序，以防止脏数据超过此阈值。

15.1.6 预读

VM 会监控文件访问模式并可能尝试执行预读。预读会在尚未请求页前预先将其从文件系统读取到页缓存中。使用此功能可以使得提交的 I/O 请求更少但更大（从而提高效率）。并可使 I/O 以管道形式执行（即在运行应用程序的同时执行 I/O）。

15.1.7 VFS 缓存

15.1.7.1 Inode 缓存

这是每个文件系统 inode 结构的内存内缓存。包含文件大小、权限和所有权以及指向文件数据的指针等属性。

15.1.7.2 目录项缓存

这是系统中目录项的内存内缓存。包含名称（文件名）、文件引用的 inode，以及子项。遍历目录结构及按名称访问文件时会使用此缓存。

15.2 减少内存用量

15.2.1 减少 malloc（匿名）用量

与旧版本相比，SUSE Linux Enterprise Desktop 15 SP7 上运行的应用程序可分配更多内存。这是因为 `glibc` 在分配用户空间内存时会更改其默认行为。有关这些参数的说明，请访问 https://www.gnu.org/s/libc/manual/html_node/Malloc-Tunable-Parameters.html 。

要恢复与旧版本类似的行为，应将 M_MMAP_THRESHOLD 设置为 128*1024。为此，可以从应用程序调用 mallopt()，或者在运行应用程序之前设置 `MALLOC_MMAP_THRESHOLD_` 环境变量。

15.2.2 减少内核内存开销

出现内存不足情况时，系统会自动清理可回收的内核内存（上述缓存）。其他大部分内核内存无法轻松缩减，而是为内核提供一个工作负载属性。

如果降低用户空间工作负载的要求，可以减少内核内存使用量（减少进程、减少打开的文件和套接字，等等）。

15.2.3 内存控制器（内存 cgroup）

如果不需要内存 cgroup 功能，可以通过在内核命令行上传递 cgroup_disable=memory 将其关闭，这可以稍微减少内核的内存消耗。还可以略微改善性能，这是因为当内存 cgroup 可用时，即使未配置任何内存 cgroup，也会产生少量的统计开销。

15.3 虚拟内存管理器 (VM) 可调参数

微调 VM 时应知悉，某些更改需要一定时间才会影响工作负载和完全见效。如果工作负载在一天中都有变化，则其行为在不同的时间可能会有所不同。在某些条件下可提高吞吐量的更改，在其他条件下可能反而会降低吞吐量。

15.3.1 回收比率

`/proc/sys/vm/swappiness`

此项控制用于定义内核换出匿名内存的主动程度（相对于页缓存和其他缓存）。增加此值会增加交换量。默认值为 `60`。

交换 I/O 的效率往往比其他 I/O 要低得多。但是，访问某些页缓存页的频率却比不常用匿名内存要高得多。应针对此点进行适当的平衡。

如果在性能下降期间观察到有交换活动，或许应考虑减小此参数。如果出现大量的 I/O 活动并且系统中的页缓存量相当小，或者正在运行大型休眠应用程序，提高此值可能会改善性能。

换出的数据越多，系统在必要情况下重新换入数据所需的时间就越长。

`/proc/sys/vm/vfs_cache_pressure`

此变量用于控制内核回收用于缓存 VFS 缓存的内存的趋势（相对于页缓存和交换）。增加此值会提高回收 VFS 缓存的速率。

想知道何时应对此值进行更改比较困难，只能通过试验来判断。**`slabtop`** 命令（`procps` 软件包的一部分）显示内核所使用的排名靠前的内存对象。vfs 缓存是“dentry”和“*_inode_cache”对象。如果这些对象消耗的内存量相对于页缓存而言很大，或许应尝试增加压力。此外，还可能有助于减少交换。默认值为 `100`。

`/proc/sys/vm/min_free_kbytes`

此参数用于控制为特殊预留用途而保留的空闲内存量，这些特殊预留包括“原子性”分配（即那些无法等待内存回收的分配操作）。除非仔细微调了系统的内存用量，否则通常不应降低此参数（通常用于嵌入式应用程序，而非服务器应用程序）。如果日志中频繁出现“页分配失败”消息和堆栈跟踪，可以持续增加 min_free_kbytes，直到错误消失。如果这些消息不常出现，则无需担心。默认值取决于 RAM 量。

`/proc/sys/vm/watermark_scale_factor`

总体而言，可用内存分为高、低、最低水平。如果达到低水平，则 **`kswapd`** 会唤醒以在后台回收内存。在可用内存达到高水平之前，它会一直处于唤醒状态。当达到最低水平时，应用程序将会卡住并回收内存。

`watermark_scale_factor` 定义在唤醒 kswapd 之前节点/系统中剩余的内存量，以及在 kswapd 回到休眠状态前需要释放多少内存。单位为万分之几。默认值 10 表示水平间的差距是节点/系统中可用内存的 0.1%。最大值为 1000，或内存的 10%。

更改此参数可能会对经常停滞在直接回收状态的工作负载（由 `/proc/vmstat` 中的 `allocstall` 判断）有益。同样，如果 **`kswapd`** 提前休眠（由 `kswapd_low_wmark_hit_quickly` 判断），可能表示为了避免停滞而保留可用的页数太小。

15.3.2 写回参数

从 SUSE Linux Enterprise Desktop 10 开始，写回行为有一个重要变更，即基于文件的 mmap() 内存发生修改后会立即被视为脏内存（可以写回）。而在以前，仅当此内存已取消映射后执行 msync() 系统调用时或处于高内存压力下，才可进行写回。

某些应用程序并不希望为 mmap 修改执行此类写回行为，因而可能会使性能下降。增加写回比率和次数可以改善这类性能下降。

`/proc/sys/vm/dirty_background_ratio`

这是总可用内存量与可回收内存量的百分比。当脏页缓存量超过此百分比时，写回线程将开始写回脏内存。默认值为 `10` (%)。

`/proc/sys/vm/dirty_background_bytes`

此参数表示当脏内存量达到该值时，后台内核刷新线程会开始执行写回操作。`dirty_background_bytes` 是 `dirty_background_ratio` 的对应参数。如果设置其中的一个，则另一个会被自动读作 `0`。

`/proc/sys/vm/dirty_ratio`

与 `dirty_background_ratio` 类似的百分比值。如果超过此值，想要写入页缓存的应用程序将被阻止，并等待内核后台刷新程序线程减少脏内存量。默认值为 `20` (%)。

`/proc/sys/vm/dirty_bytes`

此文件控制的可调参数与 `dirty_ratio` 相同，只不过其值为以字节为单位的脏内存量，而不是可回收内存的百分比。由于 `dirty_ratio` 和 `dirty_bytes` 控制的是相同的可调参数，如果设置其中的一个，另一个就会被自动读作 `0`。`dirty_bytes` 允许的最小值为两页（以字节为单位）；任何低于此限制的值都将被忽略，并将保留旧配置。

`/proc/sys/vm/dirty_expire_centisecs`

如果数据在内存中保持脏状态的时间超过此间隔，下次唤醒刷新程序线程时会将该数据写出。失效时间根据文件 inode 的修改时间计算。因此，超过该间隔时，同一文件中的多个脏页会被全部写入。

`dirty_background_ratio` 和 `dirty_ratio` 共同决定页缓存写回行为。如果增加这些值，会有更多脏内存在系统中保留更长时间。系统中允许的脏内存越多，通过避免写回 I/O 和提交更佳的 I/O 模式来提高吞吐量的可能性就越大。然而，过多的脏内存可能会在以下两种情况下造成不良影响：一是需要回收内存时（会增加延迟）；二是在数据完整性节点（即“同步点”），需要将脏内存写回磁盘时。

15.3.3 SUSE Linux Enterprise 12 与 SUSE Linux Enterprise 11 之间的 I/O 写入时间差异

系统必须限制系统内存中包含的需要写入磁盘的基于文件数据百分比。这保证了系统始终可以分配必要的数据结构来完成 I/O。任何时候允许处于脏状态且需要写入磁盘的最大内存量由 `vm.dirty_ratio`（`/proc/sys/vm/dirty_ratio`）控制。默认值为：

```
SLE-11-SP3:     vm.dirty_ratio = 40
SLE-12:         vm.dirty_ratio = 20
```

在 SUSE Linux Enterprise 12 中使用较低比率的主要优点是，在内存较低的情况下可以更快地完成页回收和分配，因为快速发现和丢弃旧干净页的概率较高。其次，如果系统上的所有数据都必须同步，则默认情况下，在 SUSE Linux Enterprise 12 上完成操作所需的时间比在 SUSE Linux Enterprise 11 SP3 上要少。大多数工作负载察觉不到此变化，因为应用程序会使用 `fsync()` 同步数据，或者数据变脏的速度还不够快，未达到限制。

但还是存在一些例外情况。如果您的应用程序受此影响，可能会在写入期间意外卡住。要证明应用程序是否受到脏数据率限制的影响，请监控 `/proc/PID_OF_APPLICATION/stack`，可以看到，应用程序在 `balance_dirty_pages_ratelimited` 中花费了大量时间。如果观察到这种情况，而且这造成了问题，请将 `vm.dirty_ratio` 值增至 40，以恢复 SUSE Linux Enterprise 11 SP3 行为。

重要

无论设置为何，总体 I/O 吞吐量都是相同的。唯一的区别仅在于 I/O 排入队列的时间。

下列示例使用 **dd** 以异步方式将 30% 的内存写入磁盘，此操作刚好受到 `vm.dirty_ratio` 中更改的影响：

```
# MEMTOTAL_MBYTES=`free -m | grep Mem: | awk '{print $2}'`
# sysctl vm.dirty_ratio=40
# dd if=/dev/zero of=zerofile ibs=1048576 count=$((MEMTOTAL_MBYTES*30/100))
2507145216 bytes (2.5 GB) copied, 8.00153 s, 313 MB/s
# sysctl vm.dirty_ratio=20
dd if=/dev/zero of=zerofile ibs=1048576 count=$((MEMTOTAL_MBYTES*30/100))
2507145216 bytes (2.5 GB) copied, 10.1593 s, 247 MB/s
```

该参数会影响完成命令所需的时间，以及设备的外显写入速度。如果设置 `dirty_ratio=40`，内核会缓存更多数据并在后台写入磁盘。在这两种情况下，I/O 的速度是相同的。下面演示了在退出之前 **dd** 同步数据时的结果：

```
# sysctl vm.dirty_ratio=40
# dd if=/dev/zero of=zerofile ibs=1048576 count=$((MEMTOTAL_MBYTES*30/100))
 conv=fdatasync
2507145216 bytes (2.5 GB) copied, 21.0663 s, 119 MB/s
# sysctl vm.dirty_ratio=20
# dd if=/dev/zero of=zerofile ibs=1048576 count=$((MEMTOTAL_MBYTES*30/100))
 conv=fdatasync
2507145216 bytes (2.5 GB) copied, 21.7286 s, 115 MB/s
```

如上所示，`dirty_ratio` 在此处几乎不会造成任何影响，且在命令的自然变化范围内。因此，`dirty_ratio` 不会直接影响 I/O 性能，但它可能会影响在未同步的情况下以异步方式写入数据的工作负载的外显性能。

15.3.4 预读参数

`/sys/block/<bdev>/queue/read_ahead_kb`

如果有一个或多个进程按顺序读取某个文件，内核会提前读取（预读）某些数据，以减少进程等待数据可用的时间。实际预读的数据量会根据 I/O 的**顺序性**程度动态计算。此参数用于设置内核预读单个文件的最大数据量。如果您发现从文件进行大量有序读取的速度不够快，可以尝试增加此值。将其设置得过高可能会导致预读抖动（即用于预读的页缓存在被使用前就被回收），或者因大量无用的 I/O 操作而导致系统性能下降。默认值为 `512` (KB)。

15.3.5 透明大页参数

利用透明大页 (THP)，可以通过进程按需动态分配大页，或者通过 **khugepaged** 内核线程将分配推迟到以后进行。此方法不同于使用 `hugetlbfs` 手动管理大页的分配和使用。THP 对于采用连续内存访问模式的工作负载很有用。在运行采用连续内存访问模式的合成工作负载时，能够发现页错误减少了 1000 倍。

在某些情况下可能不需要 THP。由于内存用量过大，采用稀疏内存访问模式的工作负载不适合使用 THP。例如，出错时可能要使用 2 MB 内存而不是 4 KB 来处理每个错误，最终导致提前回收页。在低于 SUSE Linux Enterprise 12 SP2 的版本上，尝试分配 THP 时应用程序可能会长时间处于停滞状态，因此会频繁出现禁用 THP 建议。对于 SUSE Linux Enterprise 12 SP3 和更高版本应重新评估此类建议。

可以通过 `transparent_hugepage=` 内核参数或 sysfs 配置 THP 的行为。例如，可以通过添加内核参数 `transparent_hugepage=never`，重构建 grub2 配置，然后重引导来禁用 THP。使用以下命令校验是否已禁用 THP：

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

如已禁用，方括号中会显示值 `never`，如以上示例中所示。如果指定值 `always`，会强制在出错时尝试使用 THP，但如果分配失败，则会遵从 **khugepaged**。如果指定值 `madvise`，则只会为应用程序明确指定的地址空间分配 THP。

`/sys/kernel/mm/transparent_hugepage/defrag`

此参数用于控制分配 THP 时应用程序所承担的工作量。在支持 THP 的 SUSE Linux Enterprise 12 SP1 和更低版本中，`always` 是默认值。如果 THP 不可用，应用程序会尝试对内存进行碎片整理。如果内存已碎片化并且 THP 不可用，应用程序中有可能会发生大量卡顿现象。

值 `madvise` 表示仅当应用程序明确请求时，THP 分配请求才会进行碎片整理。这是 SUSE Linux Enterprise 12 SP2 和更高版本的默认值。

`defer` 仅适用于 SUSE Linux Enterprise 12 SP2 和更高版本。如果 THP 不可用，应用程序会回退为使用小页。它会唤醒 **kswapd** 和 **kcompactd** 内核线程以在后台对内存进行碎片整理，稍后再通过 **khugepaged** 分配 THP。

最后一个选项 `never` 会在 THP 不可用的情况下使用小页，但不会执行任何其他操作。

15.3.6 khugepaged 参数

当 `transparent_hugepage` 设置为 `always` 或 `madvise` 时，khugepaged 会自动启动；当其设置为 `never` 时，khugepaged 会自动关闭。通常，khugepaged 以较低的频率运行，但可以微调行为。

`/sys/kernel/mm/transparent_hugepage/khugepaged/defrag`

0 值会禁用 **khugepaged**，虽然在出错时仍可使用 THP。对于受益于 THP，但无法容忍在 **khugepaged** 尝试更新应用程序内存用量时发生的停滞现象的延迟敏感型应用程序而言，禁用 khugepaged 可能十分重要。

`/sys/kernel/mm/transparent_hugepage/khugepaged/pages_to_scan`

此参数用于控制 **khugepaged** 在单轮中扫描的页数。扫描将会识别可重新分配为 THP 的小页。增加此值可更快地在后台分配 THP，但代价是 CPU 用量升高。

`/sys/kernel/mm/transparent_hugepage/khugepaged/scan_sleep_millisecs`

完成每轮后，**khugepaged** 将按此参数指定的短暂间隔休眠一段时间，以限制 CPU 用量。减小此值可更快地在后台分配 THP，但代价是 CPU 用量升高。指定 0 值将强制继续扫描。

`/sys/kernel/mm/transparent_hugepage/khugepaged/alloc_sleep_millisecs`

此参数用于控制当 **khugepaged** 无法在后台分配 TPH 时要休眠多久，以等待 **kswapd** 和 **kcompactd** 采取措施。

khugepaged 的其余参数极少用于性能微调，但 `/usr/src/linux/Documentation/vm/transhuge.txt` 中仍然全面阐述了这些参数

15.3.7 其他 VM 参数

有关 VM 可调参数的完整列表，请参见 `/usr/src/linux/Documentation/sysctl/vm.txt`（安装 `kernel-source` 软件包后会提供此文件）。

15.4 监控 VM 行为

一些可帮助监控 VM 行为的简单工具：

1. vmstat：此工具可让用户清楚地了解 VM 正在执行的操作。有关详细信息，请参见第 2.1.1 节 “**vmstat**”。
2. `/proc/meminfo`：此文件提供内存使用位置的明细。有关详细信息，请参见第 2.4.2 节 “内存使用详情：/proc/meminfo”。

3. **slabtop**：该工具提供有关内核 slab 内存使用情况的详细信息。buffer_head、dentry、inode_cache、ext3_inode_cache 等是主要的缓存。软件包 procps 中提供了此命令。

4. /proc/vmstat：此文件提供内部 VM 行为的明细。所包含的信息与相应实现相关，不一定始终提供。某些信息会复制到 /proc/meminfo 中，其他信息可由实用程序以易于阅读的方式呈现。为实现最大效用，需要监控此文件一段时间，以观察变化比率。很难从其他源派生的最重要信息片段如下：

   pgscan_kswapd_*, pgsteal_kswapd_*

   这些片段分别报告自系统启动以来 **kswapd** 扫描并回收的页数。这些值的比率可解释为回收效率，低效率意味着系统正在奋力回收内存，并可能出现震荡。通常不必关注此处的轻量型活动。

   pgscan_direct_*, pgsteal_direct_*

   这些片段分别报告应用程序直接扫描并回收的页数。此数字与 allocstall 计数器的增加相关联。此数字比 **kswapd** 活动更重要，因为这些事件指示进程处于停滞状态。此处的高负荷活动结合 **kswapd** 以及较高的 pgpgin、pgpout 比率和/或较高的 pswapin 或 pswpout 比率，表示系统正在经历严重的震荡。
   可以使用跟踪点来获取详细信息。

   thp_fault_alloc, thp_fault_fallback

   这些计数器对应于应用程序直接分配的 THP 数目，以及 THP 不可用的次数和使用小页的次数。一般而言，除非应用程序对 TLB 压力敏感，否则较高的回退率是无害的。

   thp_collapse_alloc, thp_collapse_alloc_failed

   这些计数器对应于 **khugepaged** 分配的 THP 数目，以及 THP 不可用的次数和使用小页的次数。较高的回退率意味着系统已碎片化，即使应用程序的内存用量允许使用 THP，也不会使用 THP。对 TLB 压力比较敏感的应用程序而言，这会成为一个问题。

   compact_*_scanned, compact_stall, compact_fail, compact_success

   当启用了 THP 且系统出现内存碎片时，这些计数器可能会增大。当应用程序在分配大页 THP 时陷入停滞状态，compact_stall 计数器会递增。其余计数器则分别记录扫描的页数、内存碎片整理成功的次数以及失败的次数。

16 微调网络

网络子系统非常复杂，其微调方式在很大程度上取决于系统的使用方案以及多种外部因素，例如网络中的软件客户端或硬件组件（交换机、路由器或网关）。Linux 内核更注重的是可靠性和低延迟，而不是低开销和高吞吐量。其他设置可能会降低安全性，但可以提高性能。

16.1 可配置的内核套接字缓冲区

大多数现代网络依赖于 TCP/IP 协议以及一个套接字接口来实现通讯；有关 TCP/IP 的详细信息，请参见《管理指南》, 第 23 章 “基本网络知识”。Linux 内核通过套接字缓冲区中的套接字接口来处理它接收或发送的数据。这些内核套接字缓冲区是可调的。

重要：TCP 自动微调

从内核版本 2.6.17 开始支持完全自动微调缓冲区大小（最大大小 4 MB）。这意味着，手动微调不会明显改善网络性能。通常，最好不要改动以下变量，或者，最起码要仔细检查执行微调的效果。

如果您要从旧版内核更新，建议去除手动 TCP 微调，改用自动微调功能。

`/proc` 文件系统中的特殊文件可以修改内核套接字缓冲区的大小和行为；有关 `/proc` 文件系统的一般信息，请参见第 2.6 节 “/proc 文件系统”。在以下位置查找网络相关的文件：

```
/proc/sys/net/core
/proc/sys/net/ipv4
/proc/sys/net/ipv6
```

内核文档 (`linux/Documentation/sysctl/net.txt`) 中解释了一般的 `net` 变量。`linux/Documentation/networking/ip-sysctl.txt` 和 `linux/Documentation/networking/ipvs-sysctl.txt` 中对特殊的 `ipv4` 变量进行了说明。

例如，在 `/proc` 文件系统中，可为所有协议设置“最大套接字接收缓冲区”和“最大套接字发送缓冲区”，或者仅为 TCP 协议设置这两个选项（在 `ipv4` 中），从而覆盖所有协议的设置（在 `core` 中）。

`/proc/sys/net/ipv4/tcp_moderate_rcvbuf`

如果 `/proc/sys/net/ipv4/tcp_moderate_rcvbuf` 设置为 `1`，则会激活自动微调并动态调整缓冲区大小。

`/proc/sys/net/ipv4/tcp_rmem`

可能的值有三个，分别设置每个连接的内存接收缓冲区的最小大小、初始大小和最大大小。这些值定义了实际内存用量，而不仅仅是 TCP 窗口大小。

`/proc/sys/net/ipv4/tcp_wmem`

与 `tcp_rmem` 相同，但用于设置每个连接的内存发送缓冲区。

`/proc/sys/net/core/rmem_max`

设置此项可以限制应用程序能够请求的最大接收缓冲区大小。

`/proc/sys/net/core/wmem_max`

设置此项可以限制应用程序能够请求的最大发送缓冲区大小。

可以通过 `/proc` 禁用您不需要的 TCP 功能（默认已打开所有 TCP 功能）。例如，检查以下文件：

`/proc/sys/net/ipv4/tcp_timestamps`

TCP 时戳在 RFC1323 中定义。

`/proc/sys/net/ipv4/tcp_window_scaling`

TCP 窗口缩放也在 RFC1323 中定义。

`/proc/sys/net/ipv4/tcp_sack`

选择确认 (SACKS)。

使用 **`sysctl`** 读取或写入 `/proc` 文件系统的变量。相比 **`cat`**（用于读取）和 **`echo`**（用于写入），使用 **`sysctl`** 更可取，因为它还能从 `/etc/sysctl.conf` 读取设置，这样每次重引导后，这些设置便能正确保留下来。使用 **`sysctl`** 可以轻松读取所有变量及其值；以 `root` 身份使用以下命令可以列出 TCP 相关的设置：

```
> sudo sysctl -a | grep tcp
```

注意：微调网络变量所造成的负面影响

微调网络变量可能会影响其他系统资源，例如 CPU 或内存用量。

16.2 检测网络瓶颈和分析网络流量

在开始进行网络微调之前，必须确定网络瓶颈和网络流量模式。某些工具可帮助您检测这些瓶颈。

以下工具可帮助分析网络流量：**`netstat`**、**`tcpdump`** 和 **`wireshark`**。Wireshark 是一个网络流量分析器。

16.3 Netfilter

Linux 防火墙和伪装功能由 Netfilter 内核模块提供。此模块是一个高度可配置的基于规则的框架。如果某个规则与某个数据包匹配，Netfilter 会接受或拒绝该数据包，或者根据规则的定义执行特殊操作（“目标”），例如地址转换。

Netfilter 可能会考虑很多属性。因此，定义的规则越多，包处理的持续时间就越长。另外，高级连接跟踪的开销可能相当大，因而会使总体网络速度变慢。

当内核队列已满时，将会丢弃所有新包，导致现有连接失败。在网络流量较大的情况下，“故障打开”功能可让用户暂时禁用包检查并保持连接。有关参考信息，请访问 https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/。

有关详细信息，请参见 Netfilter 和 iptables 项目的主页：https://www.netfilter.org。

16.4 使用接收包导向 (RPS) 改善网络性能

新式网络接口设备可以移动大量的包，导致主机成了实现最大性能的限制因素。为了跟上性能需求，系统必须能够将工作分发给多个 CPU 核心。

某些新式网络接口可以通过在硬件中实施多个传输队列和多个接收队列，帮助将工作分发到多个 CPU 核心。但是，还有一些网络接口仅配备了单个队列，驱动程序必须在单个序列化流中处理所有传入的包。要解决此问题，操作系统必须“并行化”数据流，以便将工作分散到多个 CPU 上处理。在 SUSE Linux Enterprise Desktop 上，此功能是通过接收包转向 (RPS) 实现的。还可以在虚拟环境中使用 RPS。

RPS 使用 IP 地址和端口号为每个数据流创建唯一的哈希。使用此哈希可确保将同一数据流的包发送到同一 CPU，从而帮助提升性能。

RPS 根据网络设备接收队列和接口配置。配置文件名符合以下模式：

```
/sys/class/net/<device>/queues/<rx-queue>/rps_cpus
```

<device> 代表网络设备，例如 eth0、eth1。<rx-queue> 代表接收队列，例如 rx-0、rx-1。

如果网络接口硬件仅支持单个接收队列，则只会存在 rx-0。如果它支持多个接收队列，则每个接收队列都有一个 rx-N 目录。

这些配置文件包含 CPU 位图的逗号分隔列表。默认情况下，所有位都设置为 0。使用此设置会禁用 RPS，因此，负责处理中断的 CPU 也会处理包队列。

要启用 RPS 并让特定的 CPU 来处理接口接收队列的包，请将这些 CPU 在位图中的位置值设置为 1。例如，要让 CPU 0-3 处理 eth0 的第一个接收队列的包，请将位位置 0-3 设置为二进制 1：00001111。然后，需将此表示法转换为十六进制 — 在本例中结果为 F。使用以下命令设置此十六进制值：

```
> sudo echo "f" > /sys/class/net/eth0/queues/rx-0/rps_cpus
```

如果您想要启用 CPU 8-15：

```
1111 1111 0000 0000 (binary)
15     15    0    0 (decimal)
F       F    0    0 (hex)
```

用于设置十六进制值 ff00 的命令为：

```
> sudo echo "ff00" > /sys/class/net/eth0/queues/rx-0/rps_cpus
```

在 NUMA 计算机上，要获得最佳性能，可将 RPS 配置为使用同一 NUMA 上的 CPU 来处理接口接收队列的中断。

在非 NUMA 计算机上，可以使用所有 CPU。如果中断率较高，排除用于处理网络接口的 CPU 可提升性能。可以通过 /proc/interrupts 确定用于网络接口的 CPU。例如：

```
> sudo cat /proc/interrupts
            CPU0       CPU1       CPU2       CPU3
...
  51:  113915241          0          0          0      Phys-fasteoi   eth0
...
```

在本例中，CPU 0 是唯一一个用于处理 eth0 中断的 CPU，因为只有 CPU0 包含非零值。

在 x86 和 AMD64/Intel 64 平台上，可以使用 **irqbalance** 将硬件中断分散到多个 CPU。有关详细信息，请参见 **man 1 irqbalance**。

VI 处理系统转储

17 跟踪工具 170

18 Kexec 和 Kdump 182

19 使用 systemd-coredump 针对应用程序崩溃进行调试 200

17 跟踪工具

SUSE Linux Enterprise Desktop 随附了多个工具，方便您获取有关系统的有用信息。您可以将这些信息用于不同的目的。例如，调试和查找程序中的问题、查明性能下降的原因，或者跟踪运行中的进程以确定它使用了哪些系统资源。大部分工具已包含在安装媒体中。某些情况下，需要通过单独下载的 SUSE Software Development Kit 安装这些工具。

注意：跟踪及性能影响

监控运行中进程的系统调用或库调用时，该进程的性能将严重下降。建议您仅当需要收集数据时才使用跟踪工具。

17.1 使用 strace 跟踪系统调用

strace 用于跟踪进程的系统调用及进程接收的信号。您既可以使用 **strace** 运行一个新命令并跟踪其系统调用，也可以将 **strace** 附加到一个已在运行的命令上进行跟踪。命令的每行输出都包含系统调用名称，后接其参数（括在括号中）及其返回值。

要运行新命令并开始跟踪其系统调用，请如常输入要监控的命令，并在命令行的开头添加 **strace**：

```
> strace ls
execve("/bin/ls", ["ls"], [/* 52 vars */]) = 0
brk(0)                                  = 0x618000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
        = 0x7f9848667000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
        = 0x7f9848666000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT \
(No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=200411, ...}) = 0
mmap(NULL, 200411, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9848635000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY)     = 3
```

```
[...]
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
= 0x7fd780f79000
write(1, "Desktop\nDocuments\nbin\ninst-sys\n", 31Desktop
Documents
bin
inst-sys
) = 31
close(1)                                = 0
munmap(0x7fd780f79000, 4096)            = 0
close(2)                                = 0
exit_group(0)                           = ?
```

要将 **strace** 附加到已运行的进程，需要指定 `-p` 并添加您要监控的进程的进程 ID (PID)：

```
> strace -p `pidof cron`
 Process 1261 attached
 restart_syscall(<... resuming interrupted call ...>) = 0
  stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0
  select(5, [4], NULL, NULL, {0, 0})      = 0 (Timeout)
  socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 5
  connect(5, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = 0
  sendto(5, "\2\0\0\0\0\0\0\5\0\0\0root\0", 17, MSG_NOSIGNAL, NULL, 0) = 17
  poll([{fd=5, events=POLLIN|POLLERR|POLLHUP}], 1, 5000) = 1 ([{fd=5,
 revents=POLLIN|POLLHUP}])
  read(5, "\2\0\0\0\1\0\0\0\5\0\0\0\2\0\0\0\0\0\0\0\0\0\0\5\0\0\0\6\0\0\0"...,
 36) = 36
  read(5, "root\0x\0root\0/root\0/bin/bash\0", 28) = 28
  close(5)                                = 0
  rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
  rt_sigaction(SIGCHLD, NULL, {0x7f772b9ea890, [], SA_RESTORER|SA_RESTART,
 0x7f772adf7880}, 8) = 0
  rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
  nanosleep({60, 0}, 0x7fff87d8c580)      = 0
  stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0
  select(5, [4], NULL, NULL, {0, 0})      = 0 (Timeout)
  socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 5
  connect(5, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = 0
  sendto(5, "\2\0\0\0\0\0\0\5\0\0\0root\0", 17, MSG_NOSIGNAL, NULL, 0) = 17
```

```
 poll([{fd=5, events=POLLIN|POLLERR|POLLHUP}], 1, 5000) = 1 ([{fd=5,
revents=POLLIN|POLLHUP}])
 read(5, "\2\0\0\0\1\0\0\0\5\0\0\0\2\0\0\0\0\0\0\0\0\0\0\5\0\0\0\6\0\0\0"...,
36) = 36
 read(5, "root\0x\0root\0/root\0/bin/bash\0", 28) = 28
 close(5)
 [...]
```

`-e` 选项可识别多个子选项和参数。例如，要跟踪打开或写入特定文件的所有尝试，请使用以下命令：

```
> strace -e trace=open,write ls ~
open("/etc/ld.so.cache", O_RDONLY)       = 3
open("/lib64/librt.so.1", O_RDONLY)      = 3
open("/lib64/libselinux.so.1", O_RDONLY) = 3
open("/lib64/libacl.so.1", O_RDONLY)     = 3
open("/lib64/libc.so.6", O_RDONLY)       = 3
open("/lib64/libpthread.so.0", O_RDONLY) = 3
[...]
open("/usr/lib/locale/cs_CZ.utf8/LC_CTYPE", O_RDONLY) = 3
open(".", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
write(1, "addressbook.db.bak\nbin\ncxoffice\n"..., 311) = 311
```

要仅跟踪网络相关的系统调用，请使用 `-e trace=network`：

```
> strace -e trace=network -p 26520
Process 26520 attached - interrupt to quit
socket(PF_NETLINK, SOCK_RAW, 0)         = 50
bind(50, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 0
getsockname(50, {sa_family=AF_NETLINK, pid=26520, groups=00000000}, \
[12]) = 0
sendto(50, "\24\0\0\0\26\0\1\3~p\315K\0\0\0\0\0\0\0\0", 20, 0,
{sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 20
[...]
```

`-c` 用于计算内核花费在每个系统调用上的时间：

```
> strace -c find /etc -name xorg.conf
/etc/X11/xorg.conf
% time     seconds  usecs/call     calls    errors syscall
```

```
------ ----------- ----------- --------- --------- ----------------
 32.38    0.000181         181         1           execve
 22.00    0.000123           0       576           getdents64
 19.50    0.000109           0       917        31 open
 19.14    0.000107           0       888           close
  4.11    0.000023           2        10           mprotect
  0.00    0.000000           0         1           write
[...]
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         3         1 futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         4           fadvise64
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.000559                  3633        33 total
```

要跟踪某个进程的所有子进程，请使用 `-f`：

```
> strace -f systemctl status apache2.service
execve("/usr/bin/systemctl", ["systemctl", "status", "apache2.service"], \
 0x7ffea44a3318 /* 56 vars */) = 0
brk(NULL)                               = 0x5560f664a000
[...]
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
 0x7f98c58a5000
mmap(NULL, 4420544, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
 0x7f98c524a000
mprotect(0x7f98c53f4000, 2097152, PROT_NONE) = 0
[...]
[pid  9130] read(0, "\342\227\217 apache2.service - The Apache"..., 8192) = 165
[pid  9130] read(0, "", 8027)           = 0
● apache2.service - The Apache Webserver227\217 apache2.service - Th"..., 193
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; vendor
 preset: disabled)
   Active: inactive (dead)
) = 193
[pid  9130] ioctl(3, SNDCTL_TMR_STOP or TCSETSW, {B38400 opost isig icanon
 echo ...}) = 0
```

```
[pid  9130] exit_group(0)               = ?
[pid  9130] +++ exited with 0 +++
<... waitid resumed>{si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9130, \
 si_uid=0, si_status=0, si_utime=0, si_stime=0}, WEXITED, NULL) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9130, si_uid=0, \
  si_status=0, si_utime=0, si_stime=0} ---
exit_group(3)                           = ?
+++ exited with 3 +++
```

如果您需要分析 **strace** 的输出，但输出消息太长，以致无法在控制台窗口中直接检查，请使用 `-o`。在这种情况下，将会隐藏不需要的消息，例如有关附加和分离进程的信息。您还可以使用 `-q` 隐藏这些消息（正常情况下会列显在标准输出中）。要在系统调用的每行开头添加时戳，请使用 `-t`：

```
> strace -t -o strace_sleep.txt sleep 1; more strace_sleep.txt
08:44:06 execve("/bin/sleep", ["sleep", "1"], [/* 81 vars */]) = 0
08:44:06 brk(0)                         = 0x606000
08:44:06 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, \
-1, 0) = 0x7f8e78cc5000
[...]
08:44:06 close(3)                       = 0
08:44:06 nanosleep({1, 0}, NULL)        = 0
08:44:07 close(1)                       = 0
08:44:07 close(2)                       = 0
08:44:07 exit_group(0)                  = ?
```

您可以控制 strace 的行为和输出格式。有关详细信息，请参见相关手册页 (man 1 strace)。

17.2 使用 ltrace 跟踪库调用

ltrace 可跟踪进程的动态库调用。其用法与 **strace** 类似，两者的大部分参数具有类似或相同的含义。默认情况下，**ltrace** 使用 `/etc/ltrace.conf` 或 `~/.ltrace.conf` 配置文件。不过，您可以使用 `-F CONFIG_FILE` 选项指定替代的配置文件。

除了跟踪库调用以外，带有 `-S` 选项的 **ltrace** 还可以跟踪系统调用：

```
> ltrace -S -o ltrace_find.txt find /etc -name \
xorg.conf; more ltrace_find.txt
```

```
SYS_brk(NULL)                                                  = 0x00628000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                           = 0x7f1327ea1000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                           = 0x7f1327ea0000
[...]
fnmatch("xorg.conf", "xorg.conf", 0)                           = 0
free(0x0062db80)                                               = <void>
__errno_location()                                             = 0x7f1327e5d698
__ctype_get_mb_cur_max(0x7fff25227af0, 8192, 0x62e020, -1, 0) = 6
__ctype_get_mb_cur_max(0x7fff25227af0, 18, 0x7f1327e5d6f0, 0x7fff25227af0,
0x62e031) = 6
__fprintf_chk(0x7f1327821780, 1, 0x420cf7, 0x7fff25227af0, 0x62e031
<unfinished ...>
SYS_fstat(1, 0x7fff25227230)                                   = 0
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                           = 0x7f1327e72000
SYS_write(1, "/etc/X11/xorg.conf\n", 19)                       = 19
[...]
```

您可以使用 `-e` 选项更改受跟踪事件的类型。以下示例列显与 `fnmatch` 和 `strlen` 函数相关的库调用：

```
> ltrace -e fnmatch,strlen find /etc -name xorg.conf
[...]
fnmatch("xorg.conf", "xorg.conf", 0)             = 0
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
fnmatch("xorg.conf", "Xresources", 0)            = 1
strlen("xorg.conf.install")                      = 17
[...]
```

要仅显示特定库中包含的符号，请使用 `-l /path/to/library`：

```
> ltrace -l /lib64/librt.so.1 sleep 1
clock_gettime(1, 0x7fff4b5c34d0, 0, 0, 0)                   = 0
clock_gettime(1, 0x7fff4b5c34c0, 0xffffffffff600180, -1, 0) = 0
+++ exited (status 0) +++
```

您可以使用 `-n NUM_OF_SPACES` 将每个嵌套调用按指定的空格数缩进，以此提高输出的易读性。

17.3 使用 Valgrind 进行调试和分析

Valgrind 是一套工具，可调试和分析您的程序，以使其能够更快运行且更少出错。Valgrind 可以检测与内存管理和线程相关的问题，还可以充当用于构建新调试工具的框架。但众所周知，此工具可能会产生较高的开销，例如导致运行时间变长，或在基于计时的并发工作负载下改变正常程序行为。

17.3.1 安装

标准 SUSE Linux Enterprise Desktop 发行套件中并未随附 Valgrind。要在系统上安装 Valgrind，需要获取 SUSE Software Development Kit，然后安装该软件并运行

```
zypper install VALGRIND
```

或者浏览 SUSE Software Development Kit 目录树，找到 Valgrind 软件包并使用以下命令来安装它

```
rpm -i valgrind-VERSION_ARCHITECTURE.rpm
```

SDK 是适用于 SUSE Linux Enterprise 的模块，可以通过 SUSE Customer Center 的在线通道获得，有关细节，请参见《模块和扩展快速入门》文章。

17.3.2 支持的体系结构

SUSE Linux Enterprise Desktop 支持在以下体系结构上运行 Valgrind：

- AMD64/Intel 64
- POWER
- IBM Z

17.3.3 一般信息

Valgrind 的主要优势是它能够与现有的已编译可执行文件配合使用。您无需重新编译或修改程序即可使用它。按如下所示运行 Valgrind：

```
valgrind VALGRIND_OPTIONS your-prog YOUR-PROGRAM-OPTIONS
```

Valgrind 包含多个工具，每个工具都提供特定的功能。本节中的信息是通用信息，无论使用哪个工具都适用。最重要的配置选项是 `--tool`。此选项告知 Valgrind 要运行哪个工具。如果省略此选项，默认会选择 `memcheck`。例如，要使用 Valgrind 的 `memcheck` 工具运行 **`find ~`** `-name`.bashrc，请在命令行中输入以下命令：

`valgrind` `--tool`=memcheck find ~ -name .bashrc

下面提供了标准 Valgrind 工具列表和简要说明：

`memcheck`
: 检测内存错误。它可以帮助您微调程序，使其正常运行。

`cachegrind`
: 分析缓存预测。它可以帮助您微调程序，使其运行速度更快。

`callgrind`
: 工作方式与 `cachegrind` 类似，但还会收集其他缓存分析信息。

`exp-drd`
: 检测线程错误。它可以帮助您微调多线程程序，使其正常运行。

`helgrind`
: 另一个线程错误检测器。类似于 `exp-drd`，但使用不同的技术来分析问题。

`massif`
: 一个堆分析器。堆是用于动态内存分配的内存区域。此工具可帮助您微调程序，以减少内存用量。

`lackey`
: 用于演示工具基本功能的示例工具。

17.3.4 默认选项

Valgrind 可以在启动时读取选项。Valgrind 会检查三个位置：

1. 运行 Valgrind 的用户的主目录中的 `.valgrindrc` 文件。
2. 环境变量 `$VALGRIND_OPTS`
3. 运行 Valgrind 的当前目录中的 `.valgrindrc` 文件。

完全按照上面所列顺序分析这些资源，后面指定的选项优先于前面处理的选项。与特定 Valgrind 工具相关的选项必须使用该工具的名称加上冒号作为前缀。例如，如果您希望 `cachegrind` 始终将分析数据写入 `/tmp/cachegrind_PID.log`，请将下面一行添加到主目录中的 `.valgrindrc` 文件：

```
--cachegrind:cachegrind-out-file=/tmp/cachegrind_%p.log
```

17.3.5 Valgrind 的工作原理

Valgrind 会在可执行文件启动之前取得其控制权。它会从该可执行文件和相关的共享库中读取调试信息。可执行文件的代码将重定向到选定的 Valgrind 工具，该工具会添加自身的代码来处理调试。然后，该代码将交回给 Valgrind 核心，而执行将会继续。

例如 `memcheck` 会添加自身的代码用于检查每个内存访问。因此，程序的运行速度比在本机执行环境中要慢得多。

Valgrind 会模拟程序的每条指令。因此，它不仅会检查程序的代码，还会检查所有相关库（包括 C 库）、用于图形环境的库，等等。如果您尝试使用 Valgrind 检测错误，它还会检测关联库（例如 C、X11 或 Gtk 库）中的错误。由于您不需要频繁查看所有这些错误，因此 Valgrind 可以选择性地将这些错误消息隐藏到隐藏文件。`--gen-suppressions=yes` 告知 Valgrind 要报告这些隐藏内容，使您可以将其复制到文件中。

您应提供真实的可执行文件（机器码）作为 Valgrind 参数。例如，如果您的应用程序是通过外壳或 Perl 脚本运行的，则您可能会错误地收到与 **`/bin/sh`**（或 **`/usr/bin/perl`**）相关的错误报告。在这种情况下，可以使用 `--trace-children=yes` 来解决此问题。不过，使用可执行文件本身可避免此问题造成的任何混淆。

17.3.6 消息

Valgrind 在运行时期间会报告消息，其中包含详细错误和重要事件。以下示例解释了这些消息：

```
> valgrind --tool=memcheck find ~ -name .bashrc
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE79: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
```

```
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE82: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
[...]
==6558== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==6558== malloc/free: in use at exit: 2,228 bytes in 8 blocks.
==6558== malloc/free: 235 allocs, 227 frees, 489,675 bytes allocated.
==6558== For counts of detected errors, rerun with: -v
==6558== searching for pointers to 8 not-freed blocks.
==6558== checked 122,584 bytes.
==6558==
==6558== LEAK SUMMARY:
==6558==    definitely lost: 0 bytes in 0 blocks.
==6558==      possibly lost: 0 bytes in 0 blocks.
==6558==    still reachable: 2,228 bytes in 8 blocks.
==6558==         suppressed: 0 bytes in 0 blocks.
==6558== Rerun with --leak-check=full to see details of leaked memory.
```

==6558== 引入 Valgrind 的消息，并包含进程 ID 编号 (PID)。您可以轻松将 Valgrind 的消息与程序本身的输出区分开来，并确定哪些消息属于特定的进程。

要使 Valgrind 的消息变得更详细，请使用 `-v`，甚至可以使用 `-v -v`。

您可以让 Valgrind 将其消息发送到三个不同的位置：

1. 默认情况下，Valgrind 会将其消息发送到文件描述符 2，即标准错误输出。您可以使用 `--log-fd=FILE_DESCRIPTOR_NUMBER` 选项，告知 Valgrind 将其消息发送到任何其他文件描述符。

2. 第二种方法（也是更有用的方法）是使用 `--log-file=FILENAME` 将 Valgrind 的消息发送到相应的文件。此选项接受多个变量，例如，将 `%p` 替换为当前所分析进程的 PID。这样，您就可以根据不同文件的 PID 向其发送消息。`%q{env_var}` 会被替换为相关 `env_var` 环境变量的值。

   以下示例检查重启动 Apache Web 服务器期间可能出现的内存错误，同时跟踪子进程，并将详细的 Valgrind 消息写入到按当前进程 PID 区分的不同文件：

   ```
   > valgrind -v --tool=memcheck --trace-children=yes \
   --log-file=valgrind_pid_%p.log systemctl restart apache2.service
   ```

此进程在测试系统中创建了 52 个日志文件，花费了 75 秒的时间，而在不使用 Valgrind 的正常情况下只需 7 秒来运行 **`sudo systemctl restart apache2.service`**，运行时间大约多了 10 倍。

```
> ls -1 valgrind_pid_*log
valgrind_pid_11780.log
valgrind_pid_11782.log
valgrind_pid_11783.log
[...]
valgrind_pid_11860.log
valgrind_pid_11862.log
valgrind_pid_11863.log
```

3. 您还可能倾向于通过网络发送 Valgrind 的消息。需要使用 `--log-socket=AA.BB.CC.DD:PORT_NUM` 选项指定网络套接字的 `aa.bb.cc.dd` IP 地址和 `port_num` 端口号。如果您省略端口号，将使用 1500。
   如果远程计算机上没有任何应用程序能够接收 Valgrind 的消息，将这些消息发送到网络套接字就没有意义。正因如此，我们还连同 Valgrind 一起随附了简单的侦听程序 **`valgrind-listener`**。此侦听程序接受通过指定端口建立的连接，并将它收到的任何内容复制到标准输出。

17.3.7 错误消息

Valgrind 会记住所有错误消息，如果检测到新错误，它会将该错误与旧错误消息进行比较。此方法可让 Valgrind 检查重复的错误消息。如果出现重复的错误，它将记录该错误，但不显示任何消息。此机制可避免几百万条重复的错误让您不堪其扰。

`-v` 选项会将所有报告的摘要（按其总数排序）添加到 Valgrind 执行输出的末尾。此外，如果检测到 1000 个不同的错误或者总共检测了 10000000 个错误，Valgrind 会停止收集错误。要取消此限制并查看所有错误消息，请使用 `--error-limit=no`。

某些错误会导致其他错误。因此，请按错误的出现顺序修复错误，并持续重新检查程序。

17.4 更多信息

- 有关与所述跟踪工具相关的选项的完整列表，请参见相应的手册页（**man 1 strace**、**man 1 ltrace** 和 **man 1 valgrind**）。
- 详细讲解 Valgrind 的高级用法超出了本文档的范畴。Valgrind 有详尽的文档说明，可参考《Valgrind User Manual》（Valgrind 用户手册）(https://valgrind.org/docs/manual/manual.html)。如果您需要有关 Valgrind 或其标准工具用法和用途的更高级信息，请务必阅读这些页面。

18 Kexec 和 Kdump

Kexec 是一种可以从当前运行的内核引导进入另一内核的工具。您可以在不执行任何硬件初始化的情况下加快系统重引导速度。您还可以准备好系统，以便在系统崩溃的情况下引导进入另一内核。

18.1 简介

使用 Kexec 可将运行中的内核替换为另一内核，而无需硬性重引导。该工具非常有用，原因如下：

- 加快系统重引导速度
  如果您需要频繁重引导系统，Kexec 可以帮您节省大量时间。
- 避免不可靠的固件和硬件
  计算机硬件非常复杂，在系统启动期间可能会出现严重问题。您并不总能立即更换掉不可靠的硬件。Kexec 可将内核引导到已初始化硬件的受控环境。这样，就可以最大程度地减少系统启动失败的风险。
- 保存崩溃内核的转储
  Kexec 可以保留物理内存的内容。在**生产**内核发生故障后，**捕获**内核（在预留内存范围内运行的附加内核）会保存有故障内核的状态。保存的映像可帮助您进行后续分析。
- 无需 GRUB 2 配置即可引导
  当系统使用 Kexec 引导内核时，会跳过引导加载程序阶段。常规引导过程可能会由于引导加载程序配置中的错误而失败。使用 Kexec，您就不需要考虑引导加载程序配置是否有效。

18.2 所需的软件包

要在 SUSE® Linux Enterprise Desktop 上使用 Kexec 来加速重引导或避免潜在的硬件问题，请确保已安装 `kexec-tools` 软件包。该软件包包含名为 **`kexec-bootloader`** 的脚本，该脚本会读取引导加载程序配置，并使用与常规引导加载程序相同的内核选项来运行 Kexec。

要设置一个可在发生内核崩溃时帮助您获取调试信息的环境，请确保已安装 `makedumpfile` 软件包。

在 SUSE Linux Enterprise Desktop 中使用 Kdump 的首选方法是利用 YaST Kdump 模块。要使用 YaST 模块，请确保已安装 `yast2-kdump` 软件包。

18.3 Kexec 内部结构

Kexec 中最重要的组件是 `/sbin/kexec` 命令。您可通过两种不同的方式使用 Kexec 加载内核：

- 将内核加载到某个生产内核的地址空间，以进行常规重引导：

  ```
  # kexec -l KERNEL_IMAGE
  ```

  您可以稍后使用 **kexec** `-e` 引导到此内核。

- 将内核加载到预留的内存区域：

  ```
  # kexec -p KERNEL_IMAGE
  ```

  当系统崩溃时，会自动引导此内核。

想要在系统崩溃时引导另一内核并保留生产内核的数据，您需要预留一个系统内存专用区域。生产内核永远不会加载到此区域，因为生产内核必须始终可用。此区域用于捕获内核，以便可以保留生产内核的内存页。

要预留区域，请将选项 `crashkernel` 追加到生产内核的引导命令行。要确定 `crashkernel` 的所需值，请遵循第 18.4 节 “计算 crashkernel 分配大小”中的说明操作。

这不是捕获内核的参数。捕获内核不使用 Kexec。

捕获内核会加载到预留区域，并等待内核崩溃。然后，Kdump 会尝试调用捕获内核，因为生产内核在此阶段不再可靠。这意味着，甚至 Kdump 也有可能失败。

要加载捕获内核，需要添加内核引导参数。大多数情况下，会使用初始 RAM 文件系统进行引导。您可以使用 `--initrd=FILENAME` 指定此参数。使用 `--append=CMDLINE` 可向要引导的内核的命令行追加选项。

必须添加生产内核的命令行。只需复制包含 `--append="$(cat /proc/cmdline)"` 的命令行，或使用 `--append="$(cat /proc/cmdline) more_options"` 添加更多选项即可。

例如，要使用当前正在运行的生产内核的命令行和 `/boot/initrd` 文件加载 `/boot/vmlinuz-6.4.0-150700.38-default` 内核映像，请运行以下命令：

```
#  kexec -l /boot/vmlinuz-6.4.0-150700.38-default \
 --append="$(cat /proc/cmdline)" --initrd=/boot/initrd
```

您随时都可以卸载先前加载的内核。要卸载通过 `-l` 选项加载的内核，请使用 **kexec** `-u` 命令。要卸载通过 `-p` 选项加载的崩溃内核，请使用 **kexec** `-p -u` 命令。

18.4 计算 crashkernel 分配大小

要搭配捕获内核使用 Kexec 以及以任何方式使用 Kdump，需要为捕获内核分配 RAM。分配大小取决于计算机的预期硬件配置，因此您需要指定此配置。

分配大小还取决于计算机的硬件体系结构。请确保遵循适合您系统体系结构的过程。

过程 18.1： AMD64/INTEL 64 上的分配大小

1. 要获取计算机的基值，请运行以下命令：

   ```
   # kdumptool calibrate
   Total: 49074
   Low: 72
   High: 180
   MinLow: 72
   MaxLow: 3085
   MinHigh: 0
   MaxHigh: 45824
   ```

   所有值均以 MB 为单位。

2. 请记下 `Low` 和 `High` 的值。

注意：Low 和 High 值的含义

在 AMD64/Intel 64 计算机上，`High` 值表示所有可用内存的内存预留量。`Low` 值表示 DMA32 区域中的内存预留量，即，即 4GB 以内的全部内存。

SIZE_LOW 是仅支持 32 位的设备所需的内存量。内核会为 DMA32 回弹缓冲区分配 64M。如果您的服务器不包含任何仅支持 32 位的设备，为 `SIZE_LOW` 指定 72M 的默认分配大小应可确保一切正常。一种可能的例外情况是，在 NUMA 计算机上，似乎需要更多的 `Low` 内存。可以使用 `numa=off` 引导 Kdump 内核，以确保常规内核分配不使用 `Low` 内存。

3. 根据附加到计算机的 LUN 内核路径（存储设备的路径）数量调整上一步骤中的 `High` 值。可使用以下公式计算合理值（以 MB 为单位）：

```
SIZE_HIGH = RECOMMENDATION + (LUNs / 2)
```

此公式中使用了以下参数：

- **SIZE_HIGH：** `High` 的结果值。
- **RECOMMENDATION：** **`kdumptool calibrate`** 为 `High` 推荐的值。
- **LUNs：** 预期要在计算机上创建的最大 LUN 内核路径数。请从此数字中排除多路径设备，因为会忽略这些设备。要获取系统上**当前**可用的 LUN 数量，请运行以下命令：

```
>  cat /proc/scsi/scsi | grep Lun | wc -l
```

4. 如果设备驱动程序在 DMA32 区域中预留了大量的内存，则还需要调整 `Low` 值。但是，没有任何简单公式可以计算这些值。因此，确定适当大小可能需要经历一个反复试验的过程。
   一开始请使用 **`kdumptool calibrate`** 推荐的 `Low` 值。

5. 现在，需要在正确的位置设置值。

如果您直接更改内核命令行

请将以下内核选项追加到引导加载程序配置：

```
crashkernel=SIZE_HIGH,high crashkernel=SIZE_LOW,low
```

请将占位符 SIZE_HIGH 和 SIZE_LOW 替换为前面步骤中的相应值，并追加字母 M（表示 MB）。
例如，以下设置有效：

```
crashkernel=36M,high crashkernel=72M,low
```

如果您使用的是 YaST GUI：

将 Kdump 内存下限设置为已确定的 Low 值。
将 Kdump 内存上限设置为已确定的 High 值。

如果您使用的是 YaST 命令行界面：

使用以下命令：

```
# yast kdump startup enable alloc_mem=LOW,HIGH
```

请将 LOW 替换为已确定的 Low 值。请将 HIGH 替换为已确定的 HIGH 值。

提示：排除 IBM Z 上未用和非活动的 CCW 设备

根据可用设备的数量，crashkernel 内核参数指定的计算内存量可能并不足够。您可以不增大该值，而是限制内核可见的设备数量。这可以降低 **crashkernel** 设置所需的内存量。

1. 要忽略设备，可以运行 **cio_ignore** 工具生成相应的节来忽略所有设备（当前活动的或使用中的设备除外）。

```
> sudo cio_ignore -u -k
cio_ignore=all,!da5d,!f500-f502
```

当您运行 **cio_ignore -u -k** 时，该黑名单将会激活，并立即替换任何现有黑名单。不会清除未使用的设备，因此它们仍会显示在通道子系统中。但如果添加新通道设备（通过 z/VM 下的 CP ATTACH，或 LPAR 中的动态 I/O 配置更改），会将这

些设备视为已列入黑名单。要防止出现这种情况，请先运行 **sudo cio_ignore -l** 来保留原始设置，然后在运行 **cio_ignore -u -k** 后还原到该状态。也可将生成的节添加到常规内核引导参数。

2. 现在，将包含上述节的 cio_ignore 内核参数添加到 /etc/sysconfig/kdump 中的 KDUMP_CMDLINE_APPEND，例如：

```
KDUMP_COMMANDLINE_APPEND="cio_ignore=all,!da5d,!f500-f502"
```

3. 通过重启动 kdump 激活该设置：

```
systemctl restart kdump.service
```

18.5 Kexec 基本用法

要使用 Kexec，请确保相应的服务已启用并正在运行：

- 确保在启动系统时加载 Kexec 服务：

```
> sudo systemctl enable kexec-load.service
```

- 确保 Kexec 服务正在运行：

```
> sudo systemctl start kexec-load.service
```

要校验您的 Kexec 环境是否正常工作，请尝试使用 Kexec 重引导到新的内核。确保当前没有任何用户登录，且没有任何重要服务正在系统上运行。然后运行以下命令：

```
systemctl kexec
```

先前已加载到旧内核的地址空间中的新内核会重新写入，并立即接管控制权。它会如常显示启动消息。当新内核引导时，会跳过所有硬件和固件检查。请确保未显示任何警告消息。

提示：在 reboot 命令中使用 Kexec

要使 **reboot** 使用 Kexec 而不是执行常规重引导，请运行以下命令：

```
ln -s /usr/lib/systemd/system/kexec.target /etc/systemd/system/
reboot.target
```

随时可以通过删除 `etc/systemd/system/reboot.target` 来还原此设置。

18.6 如何为例程重引导配置 Kexec

Kexec 通常用于频繁重引导。例如，如果运行整个硬件检测例程需要花费很长时间或者启动不可靠，则可以使用 Kexec。

使用 Kexec 重引导系统时，不会使用固件和引导加载程序。在计算机执行硬性重引导之前，对引导加载程序配置所做的任何更改都会被忽略。

18.7 Kdump 基本配置

可以使用 Kdump 保存内核转储。如果内核崩溃，将已崩溃环境的内存映像复制到文件系统会很有用。然后，您可以调试转储文件，以找出内核崩溃的原因。这称为“核心转储”。

Kdump 的工作方式与 Kexec 类似（请参见第 18 章 “Kexec 和 Kdump”）。运行中的生产内核崩溃后，会执行捕获内核。区别在于，Kexec 将生产内核替换成了捕获内核。使用 Kdump 时，仍可以访问已崩溃生产内核的内存空间。可以在 Kdump 内核的环境中保存已崩溃内核的内存快照。

提示：通过网络转储

在本地存储空间有限的环境中，需要设置通过网络进行内核转储。Kdump 支持配置指定的网络接口，并通过 `initrd` 启动该接口。LAN 和 VLAN 接口均受支持。通过 YaST 或者在 `/etc/sysconfig/kdump` 文件中使用 `KDUMP_NETCONFIG` 选项来指定网络接口和模式（DHCP 或静态）。

重要：必须在配置期间挂载 Kdump 的目标文件系统

配置 Kdump 时，可以指定用于保存转储映像的位置（默认为 `/var/crash`）。必须在配置 Kdump 期间挂载此位置，否则配置会失败。

18.7.1 Kdump 的手动配置

Kdump 从 `/etc/sysconfig/kdump` 文件读取其配置。Kdump 的默认配置足以确保其可在您的系统上正常工作。要以默认设置使用 Kdump，请遵循以下步骤：

1. 遵循第 18.4 节 “计算 crashkernel 分配大小”中的说明确定 Kdump 所需的内存量。确保设置内核参数 `crashkernel`。
2. 重新启动计算机。
3. 启用 Kdump 服务：

```
# systemctl enable kdump
```

4. 可以编辑 `/etc/sysconfig/kdump` 中的选项。查看注释有助于了解各个选项的含义。
5. 使用 **`sudo systemctl start kdump`** 执行一次 init 脚本，或重引导系统。

使用默认值配置 Kdump 后，检查它是否按预期工作。确保当前没有任何用户登录，且没有任何重要服务正在系统上运行。然后遵循以下步骤：

1. 使用 **`systemctl isolate rescue.target`** 切换到救援目标
2. 重启动 Kdump 服务：

```
# systemctl start kdump
```

3. 使用以下命令卸载除根文件系统以外的所有磁盘文件系统：

```
# umount -a
```

4. 以只读模式重新挂载根文件系统：

```
# mount -o remount,ro /
```

5. 通过 Magic SysRq 组合键的 `procfs` 接口触发“内核崩溃”：

```
# echo c > /proc/sysrq-trigger
```

重要：内核转储大小

`KDUMP_KEEP_OLD_DUMPS` 选项控制保留的内核转储数（默认为 5 个）。如果不压缩，转储大小最大可为物理内存或 RAM 的大小。请确保 `/var` 分区上有足够的空间。

捕获内核将会引导，已崩溃内核的内存快照将保存到文件系统。保存路径由 `KDUMP_SAVEDIR` 选项指定，默认为 `/var/crash`。如果 `KDUMP_IMMEDIATE_REBOOT` 设置为 `yes`，系统会自动重引导生产内核。登录并检查是否已在 `/var/crash` 下创建了转储。

18.7.2 YaST 配置

要使用 YaST 配置 Kdump，需要安装 `yast2-kdump` 软件包。然后在 YaST 控制中心的系统类别中启动内核 Kdump 模块，或者以 `root` 身份在命令行中输入 **`yast2 kdump`**。

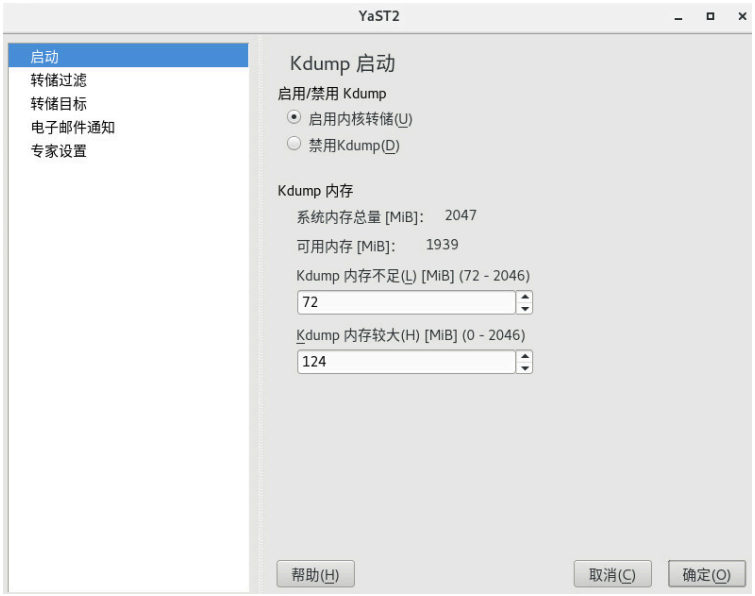

图 18.1： YAST KDUMP 模块：启动页

在启动窗口中选择启用 Kdump。

当您首次打开该窗口时，系统会自动生成 Kdump 内存的值。但是，这并不意味着这些值始终足以满足需求。要设置正确的值，请遵循第 18.4 节 “计算 crashkernel 分配大小”中的说明操作。

重要：更改硬件后再次设置 Kdump 内存值

如果您在计算机上设置了 Kdump 后又决定更改 Kdump 的可用 RAM 或硬盘容量，YaST 会继续显示并使用已过时的内存值。

要解决此问题，请根据第 18.4 节 “计算 crashkernel 分配大小”中所述再次确定所需内存。然后在 YaST 中手动设置内存。

在左窗格中单击转储过滤，然后检查要包含在转储中的页。无需包含以下内存内容即可调试内核问题：

- 以零填充的页
- 缓存页
- 用户数据页
- 可用页

在转储目标窗口中，选择转储目标的类型，以及要将转储内容保存到的 URL。如果您选择了 FTP 或 SSH 等网络协议，则还需要输入相关的访问信息。

提示：与其他应用程序共享转储目录

可以指定 Kdump 转储的保存路径，其他应用程序也可以在此路径中保存其转储。当 Kdump 清理其旧转储文件时，会忽略其他应用程序的转储文件，因而不会造成任何问题。

如果您希望 Kdump 通过电子邮件将与其相关的事件通知给您，那么在专家设置窗口中微调 Kdump 后，请在电子邮件通知窗口中填写相应信息，并单击确定确认更改。现已配置 Kdump。

18.7.3 Kdump 通过 SSH 传输

转储文件常常包含敏感数据，因此应防范未经授权透露这些数据。要允许通过不安全的网络传输此类数据，Kdump 可以使用 SSH 协议将转储文件保存到远程计算机。

1. 目标主机身份对于 Kdump 必须是已知的。需要做到这一点才能确保永远不会向欺骗者发送敏感数据。当 Kdump 生成新的 `initrd` 时，会运行 **`ssh-keygen -F`** `TARGET_HOST` 来查询目标主机的身份。仅当 `TARGET_HOST` 公共密钥已知时，才可完成此查询。实现该操作的一种简单方法是在 Kdump 主机上以 `root` 身份与 `TARGET_HOST` 建立 SSH 连接。

2. Kdump 必须能够对目标计算机进行身份验证。目前只能使用公共密钥身份验证。Kdump 默认会使用 `root` 的私用密钥，但建议您为 Kdump 创建单独的密钥。可以使用 **`ssh-keygen`** 执行此操作：

   a.
   ```
   # ssh-keygen -f ~/.ssh/kdump_key
   ```

   b. 当系统提示输入通行口令时，请按 `Enter` （即，不使用任何通行口令）。

   c. 打开 `/etc/sysconfig/kdump` 并将 `KDUMP_SSH_IDENTITY` 设置为 `kdump_key`。如果该文件不在 `~/.ssh` 下，可以使用其完整路径。

3. 设置 Kdump SSH 密钥以授权登录到远程主机。

   ```
   # ssh-copy-id -i ~/.ssh/kdump_key TARGET_HOST
   ```

4. 设置 `KDUMP_SAVEDIR`。有两个选项：

   安全文件传输协议 (SFTP)

   SFTP 是通过 SSH 传输文件的首选方法。目标主机必须启用 SFTP 子系统（SUSE Linux Enterprise Desktop 默认已启用）。示例：

   ```
   KDUMP_SAVEDIR=sftp://TARGET_HOST/path/to/dumps
   ```

   安全外壳协议 (SSH)

   其他某些发行套件使用 SSH 在目标主机上运行某些命令。SUSE Linux Enterprise Desktop 也可以使用此方法。目标主机上的 Kdump 用户必须具有一个可执行以下命令的登录外壳：**`mkdir`**、**`dd`** 和 **`mv`**。示例：

```
KDUMP_SAVEDIR=ssh://TARGET_HOST/path/to/dumps
```

5. 重启动 Kdump 服务以使用新配置。

18.8 分析崩溃转储

获取转储后，便可以开始对它进行分析了。系统提供了多个选项。

用于分析转储的初始工具为 GDB。您甚至可以在最新的环境中使用此工具，不过，它也存在一些缺点和限制：

- GDB 并非专用于调试内核转储的工具。
- GDB 不支持 32 位平台上的 ELF64 二进制文件。
- GDB 不能识别除 ELF 转储以外的其他格式（无法调试压缩的转储）。

由于这些原因，我们实施的实用程序是 **crash**。crash 能够分析崩溃转储，还能调试运行中的系统。它提供了专用于调试 Linux 内核的功能，更适合用于高级调试。

要调试 Linux 内核，还需安装其调试信息软件包。使用以下命令检查您的系统上是否已安装该软件包：

```
> zypper se kernel | grep debug
```

重要：包含调试信息的软件包的储存库

如果您为系统订阅了联机更新，便可以在 SUSE Linux Enterprise Desktop 15 SP7 的相关 `*-Debuginfo-Updates` 联机安装储存库中找到 “debuginfo” 软件包。使用 YaST 启用该储存库。

要在生成转储的计算机上的 **crash** 中打开捕获的转储，请使用如下所示的命令：

```
crash /boot/vmlinux-6.4.0-150700.38-default.gz \
/var/crash/2024-04-23-11\:17/vmcore
```

第一个参数表示内核映像。第二个参数是 Kdump 捕获的转储文件。默认可在 `/var/crash` 下找到此文件。

提示：从内核崩溃转储中获取基本信息

SUSE Linux Enterprise Desktop 随附了 **kdumpid** 实用程序（包含在同名的软件包中），用于识别未知的内核转储。可以使用此实用程序来提取基本信息，例如体系结构和内核版本。它支持 lkcd、diskdump、Kdump 文件和 ELF 转储。使用 `-v` 开关调用此实用程序时，它会尝试提取额外的信息，例如计算机类型、内核标题字符串和内核配置变种。

18.8.1 内核二进制文件格式

Linux 内核采用可执行与可链接格式 (ELF)。此文件名为 `vmlinux`，直接在编译过程中生成。并非所有引导加载程序都支持 ELF 二进制文件，尤其是在 AMD64/Intel 64 体系结构上。针对 SUSE® Linux Enterprise Desktop 支持的不同体系结构，我们提供了以下解决方案。

18.8.1.1 AMD64/Intel 64

SUSE 提供的适用于 AMD64/Intel 64 的内核软件包中包含两个内核文件：`vmlinuz` 和 `vmlinux.gz`。

- `vmlinuz`： 这是由引导加载程序执行的文件。
  Linux 内核由两个部分组成：内核本身 (`vmlinux`)，以及由引导加载程序运行的设置代码。这两个部分关联在一起创建了 `vmlinuz`（请注意字母 `z` 与 `x` 的差别）。
  在内核源树中，该文件名为 `bzImage`。
- `vmlinux.gz`： 这是可由 **crash** 和 GDB 使用的压缩 ELF 映像。在 AMD64/Intel 64 上，引导加载程序本身永远不会使用 ELF 映像。因此，只随附了一个压缩版本。

18.8.1.2 POWER

POWER 上的 `yaboot` 引导加载程序也支持加载 ELF 映像，但不支持压缩的 ELF 映像。在 POWER 内核软件包中，有一个 ELF Linux 内核文件 `vmlinux`。对于 **crash** 而言，POWER 是最简单的体系结构。

如果您决定在另一台计算机上分析转储，必须检查计算机的体系结构，以及调试所需的文件。

仅当另一台计算机运行相同体系结构的 Linux 系统时，才能在该计算机上分析转储。要检查兼容性，请在两台计算机上使用命令 **uname** `-i` 并比较输出。

如果您要在另一台计算机上分析转储，还需要获取 `kernel` 和 `kernel debug` 软件包中的适当文件。

1. 将内核转储、`/boot` 中的内核映像以及 `/usr/lib/debug/boot` 中的关联调试信息文件放到一个空目录中。
2. 此外，请将 `/lib/modules/$(uname -r)/kernel/` 中的内核模块以及 `/usr/lib/debug/lib/modules/$(uname -r)/kernel/` 中的关联调试信息复制到名为 `modules` 的子目录中。
3. 在包含转储、内核映像及其调试信息文件以及 `modules` 子目录的目录中，启动 **crash** 实用程序：

```
> crash VMLINUX-VERSION vmcore
```

不管您在哪台计算机上分析转储，crash 实用程序都会生成类似如下内容的输出：

```
> crash /boot/vmlinux-6.4.0-150700.38-default.gz \
/var/crash/2024-04-23-11\:17/vmcore
crash 7.2.1
Copyright (C) 2002-2017  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006, 2010  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006, 2011, 2012  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005, 2011  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb (GDB) 7.6
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".

      KERNEL: /boot/vmlinux-6.4.0-150700.38-default.gz
   DEBUGINFO: /usr/lib/debug/boot/vmlinux-6.4.0-150700.38-default.debug
    DUMPFILE: /var/crash/2024-04-23-11:17/vmcore
        CPUS: 2
        DATE: Thu Apr 23 13:17:01 2024
      UPTIME: 00:10:41
LOAD AVERAGE: 0.01, 0.09, 0.09
       TASKS: 42
    NODENAME: eros
     RELEASE: 6.4.0-150700.38-default
     VERSION: #1 SMP 2024-03-31 14:50:44 +0200
     MACHINE: x86_64  (2999 Mhz)
      MEMORY: 16 GB
       PANIC: "SysRq : Trigger a crashdump"
         PID: 9446
     COMMAND: "bash"
        TASK: ffff88003a57c3c0  [THREAD_INFO: ffff880037168000]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)
crash>
```

命令输出会先列显有用数据：内核崩溃时有 42 个任务在运行。崩溃的原因是 PID 为 9446 的任务调用了 SysRq 触发器。此任务是一个 Bash 进程，因为使用的 **`echo`** 是 Bash 外壳的内部命令。

`crash` 实用程序是在 GDB 的基础上构建的，可提供许多额外的命令。如果您输入不带任何参数的 **`bt`**，将列显在崩溃时正在运行任务的回溯：

```
crash> bt
PID: 9446   TASK: ffff88003a57c3c0  CPU: 1   COMMAND: "bash"
 #0 [ffff880037169db0] crash_kexec at ffffffff80268fd6
 #1 [ffff880037169e80] __handle_sysrq at ffffffff803d50ed
 #2 [ffff880037169ec0] write_sysrq_trigger at ffffffff802f6fc5
 #3 [ffff880037169ed0] proc_reg_write at ffffffff802f068b
```

```
 #4 [ffff880037169f10] vfs_write at ffffffff802b1aba
 #5 [ffff880037169f40] sys_write at ffffffff802b1c1f
 #6 [ffff880037169f80] system_call_fastpath at ffffffff8020bfbb
    RIP: 00007fa958991f60  RSP: 00007fff61330390  RFLAGS: 00010246
    RAX: 0000000000000001  RBX: ffffffff8020bfbb  RCX: 0000000000000001
    RDX: 0000000000000002  RSI: 00007fa959284000  RDI: 0000000000000001
    RBP: 0000000000000002   R8: 00007fa9592516f0   R9: 00007fa958c209c0
    R10: 00007fa958c209c0  R11: 0000000000000246  R12: 00007fa958c1f780
    R13: 00007fa959284000  R14: 0000000000000002  R15: 00000000595569d0
    ORIG_RAX: 0000000000000001  CS: 0033  SS: 002b
crash>
```

现在，发生的情况便一清二楚了：Bash 外壳的内部 **echo** 命令向 `/proc/sysrq-trigger` 发送了一个字符。相应的处理程序在识别此字符后，调用了 `crash_kexec()` 函数。此函数调用了 `panic()`，且 Kdump 保存了转储。

除了基本的 GDB 命令以及 **bt** 的扩展版本以外，crash 实用程序还定义了与 Linux 内核结构相关的其他命令。这些命令可识别 Linux 内核的内部数据结构，并以直观易懂的格式显示这些内容。例如，您可以使用 **ps** 列出崩溃时正在运行的任务。使用 **sym** 可以列出包含相应地址的所有内核符号，或查询单个符号的值。使用 **files** 可以显示进程的所有打开文件描述符。使用 **kmem** 可以显示有关内核内存用量的细节。使用 **vm** 可以检查进程的虚拟内存，甚至是在单个页映射级别。有用命令的列表很长，其中有许多命令都会接受各种各样的选项。

我们所提及的命令反映了常用 Linux 命令（例如 **ps** 和 **lsof**）的功能。要使用调试程序找出确切的事件顺序，需要了解 GDB 的用法并具备强大的调试技能。这些知识超出了本文档的说明范围。此外，您需要了解 Linux 内核。本文档的末尾提供了多个有用的参考信息来源。

18.9 Kdump 高级配置

Kdump 的配置存储在 `/etc/sysconfig/kdump` 中。您也可以使用 YaST 来配置 Kdump。您可以在 YaST 控制中心中的系统 › 内核 Kdump 下访问 Kdump 配置选项。以下 Kdump 选项可能对您有用。

可以使用 KDUMP_SAVEDIR 选项更改内核转储的目录。请注意，内核转储大小可能会很大。如果减去预估转储大小后的可用磁盘空间低于 KDUMP_FREE_DISK_SIZE 选项指定的值，Kdump 将拒绝保存转储内容。KDUMP_SAVEDIR 可以识别 URL 格式 PROTOCOL://

SPECIFICATION，其中 PROTOCOL 是 file、ftp、sftp、nfs 或 cifs 之一，并且每种协议的 specification 不尽相同。例如，要在 FTP 服务器上保存内核转储，请使用以下 URL 作为模板：ftp://username:password@ftp.example.com:123/var/crash。

内核转储很大，包含许多分析所不需要的页。使用 KDUMP_DUMPLEVEL 选项可以省略此类页。该选项可接受 0 到 31 之间的数字值。如果指定 0，转储大小是最大的。如果指定 31，将生成最小的转储。有关可能值的完整表格，请参见 **kdump** 的手册页 (**man 7 kdump**)。

有时，减小内核转储大小会有所帮助。例如，如果您要通过网络传送转储，或者需要节省转储目录的磁盘空间，就可以采用这种做法。可将 KDUMP_DUMPFORMAT 设置为 compressed 来实现此目的。**crash** 实用程序支持对压缩的转储进行动态解压缩。

重要：对 Kdump 配置文件的更改

对 /etc/sysconfig/kdump 文件进行更改后，需要运行 **systemctl restart kdump.service**。否则，要到下次重引导系统时，这些更改才会生效。

18.10 更多信息

目前还没有介绍 Kexec 和 Kdump 用法的综合性参考文档。但是，下面这些有用资源可帮助您了解某些特定方面：

- 有关 Kexec 实用程序的用法，请参见 **kexec** 的手册页 (**man 8 kexec**)。
- 您可以在 https://developer.ibm.com/technologies/linux/ 上找到有关 Kexec 的一般信息。
- 有关专用于 SUSE Linux Enterprise Desktop 的 Kdump 的更多细节，请访问 https://ftp.suse.com/pub/people/tiwai/kdump-training/kdump-training.pdf。
- 您可以在 https://lse.sourceforge.net/kdump/documentation/ols2oo5-kdump-paper.pdf 上找到 Kdump 内部机制的深入说明。

有关 **crash** 转储分析和调试工具的更多细节，请使用以下资源：

- 除 GDB 的信息页 (**info gdb**) 外，https://sourceware.org/gdb/documentation/ 上还提供了可打印的指南。
- crash 实用程序提供综合性的联机帮助。使用 **help** COMMAND 可显示 **command** 的联机帮助。
- 如果您具备所需的 Perl 技能，可以使用 Alicia 来简化调试。您可以在 https://alicia.sourceforge.net/ 上找到 crash 实用程序对应的 Perl 前端。
- 如果偏好使用 Python，则应安装 Pykdump。此软件包可帮助您通过 Python 脚本控制 GDB。
- 由 Daniel P. Bovet 和 Marco Cesati 撰写的 Understanding the Linux Kernel (ISBN 978-0-596-00565-8) 中全面概述了 Linux 内核的内部结构。

19 使用 systemd-coredump 针对应用程序崩溃进行调试

systemd-coredump 可收集并显示核心转储，用于分析应用程序崩溃问题。核心转储包含进程内存在终止时的映像。默认情况下，当某个进程（或属于应用程序的所有进程）崩溃时，此工具会将核心转储存储在 /var/lib/systemd/coredump 文件中，并将核心转储记录到 systemd 日记（如有可能，还包括回溯）。您还可以选择使用 **gdb** 或 **crash** 等其他工具检查转储文件（请参见第 18.8 节 “分析崩溃转储”）。

存储在 /var/lib/systemd/coredump 中的核心转储在三天后将被删除（请参见 /usr/lib/tmpfiles.d/systemd.conf 中的 d /var/lib/systemd/coredump 行）。

有一个选项不会存储核心转储，而是仅将其记录到日记，这样做有助于尽量减少敏感信息的收集与存储。

19.1 用法和配置

systemd-coredump 默认已启用，随时可供运行。默认配置位于 /etc/systemd/coredump.conf 中：

```
[Coredump]
#Storage=external
#Compress=yes
#ProcessSizeMax=2G
#ExternalSizeMax=2G
#JournalSizeMax=767M
#MaxUse=
#KeepFree=
```

大小单位为 B、K、M、G、T、P 和 E。ExternalSizeMax 还支持 infinity 值。

以下示例显示如何使用 Vim 进行简单的测试，方法是创建一个 segfault 来生成日记项和核心转储。

过程 19.1： 使用 VIM 创建核心转储

1. 启用 `debuginfo-pool` 和 `debuginfo-update` 储存库
2. 安装 `vim-debuginfo`
3. 启动 **`vim testfile`** 并键入几个字符
4. 获取 PID 并生成 segfault：

```
> ps ax | grep vim
2345 pts/3    S+     0:00 vim testfile

# kill -s SIGSEGV 2345
```

Vim 会发出错误消息：

```
Vim: Caught deadly signal SEGV
Vim: Finished.
Segmentation fault (core dumped)
```

5. 列出您的核心转储，然后对其进行检查：

```
# coredumpctl
TIME                          PID  UID  GID SIG PRESENT EXE
Wed 2019-11-12 11:56:47 PST 2345 1000 100 11  *       /bin/vim

# coredumpctl info
PID: 2345 (vim)
UID: 0 (root)
GID: 0 (root)
Signal: 11 (SEGV)
Timestamp: Wed 2019-11-12 11:58:05 PST
Command Line: vim testfile
Executable: /bin/vim
Control Group: /user.slice/user-1000.slice/session-1.scope
    Unit: session-1.scope
    Slice: user-1000.slice
    Session: 1
    Owner UID: 1000 (tux)
    Boot ID: b5c251b86ab34674a2222cef102c0c88
```

```
    Machine ID: b43c44a64696799b985cafd95dc1b698
    Hostname: linux-uoch
    Coredump: /var/lib/systemd/coredump/
core.vim.0.b5c251b86ab34674a2222cef102
    Message: Process 2345 (vim) of user 0 dumped core.

         Stack trace of thread 2345:
         #0  0x00007f21dd87e2a7 kill (libc.so.6)
         #1  0x000000000050cb35 may_core_dump (vim)
         #2  0x00007f21ddbfec70 __restore_rt (libpthread.so.0)
         #3  0x00007f21dd92ea33 __select (libc.so.6)
         #4  0x000000000050b4e3 RealWaitForChar (vim)
         #5  0x000000000050b86b mch_inchar (vim)
[...]
```

如果您有多个核心转储，**coredumpctl info** 会显示所有这些核心转储。可按 PID、COMM（命令）或 EXE（可执行文件的完整路径）过滤核心转储，例如，要查看 Vim 的所有核心转储，请使用以下命令：

```
# coredumpctl info /bin/vim
```

按 PID 查看单个核心转储：

```
# coredumpctl info 2345
```

将选定的核心输出到 **gdb**：

```
# coredumpctl gdb 2345
```

PRESENT 列中的星号表示存在已存储的核心转储。如果该字段为空，则表示没有已存储的核心转储，**coredumpctl** 将从日记中检索崩溃信息。可以在 /etc/systemd/coredump.conf 中使用 Storage 选项控制此行为：

- Storage=none — 在日记中记录核心转储，但不存储。这样做有助于尽量减少敏感信息的收集与存储，例如，出于遵守《通用数据保护条例》(GDPR) 的目的。
- Storage=external — 将核心存储在 /var/lib/systemd/coredump 中
- Storage=journal — 将核心存储在 systemd 日记中

将为每个核心转储调用 systemd-coredump 的新实例，因此，下一次核心转储时会应用配置更改，而无需重启动任何服务。

重启动系统后核心转储不会保留。可以使用 **coredumpctl** 永久保存核心转储。以下示例按 PID 过滤核心转储，并将核心存储在 vim.dump 中：

```
# coredumpctl -o vim.dump dump 2345
```

有关完整的命令和选项列表，请参见 **man systemd-coredump**、**man coredumpctl**、**man core** 和 **man coredump.conf**。

VII 使用精确时间协议同步时钟

20 精确时间协议 205

20 精确时间协议

对于网络环境而言，让计算机和其他设备的时钟保持同步和准确至关重要。有多种解决方案可实现同步性和准确性，例如，使用《管理指南》, 第 39 章 “使用 NTP 同步时间”中所述的广泛应用的网络时间协议 (NTP)。

精确时间协议 (PTP) 支持亚微秒级准确性，较 NTP 更优。PTP 支持分为内核和用户空间。SUSE Linux Enterprise Desktop 中的内核支持网络驱动程序提供的 PTP 时钟。

20.1 PTP 简介

PTP 管理的时钟遵循主从层次结构。从属时钟将同步到其主时钟。这一层次结构由**最佳主时钟** (BMC) 算法负责更新，该算法会针对每个时钟运行。只有一个端口的时钟可以是主时钟也可以是从属时钟。此类时钟称为**普通时钟** (OC)。具有多个端口的时钟可以在一个端口上为主时钟，在另一个端口上为从属时钟。此类时钟称为**边界时钟** (BC)。顶层的主时钟称为**超级主时钟**。超级主时钟可与全球定位系统 (GPS) 同步。这样，不同的网络便能够以高准确度实现同步。

硬件支持是 PTP 的主要优势。多种网络交换机和网络接口控制器 (NIC) 都支持 PTP。虽然可以在网络内部使用启用非 PTP 的硬件，但为所有 PTP 时钟之间的网络组件启用 PTP 硬件可实现最大的准确性。

20.1.1 PTP Linux 实现

在 SUSE Linux Enterprise Desktop 上，PTP 实现由 `linuxptp` 软件包提供。可通过 **`zypper install linuxptp`** 安装该软件包。该软件包中包含用于同步时钟的 **`ptp4l`** 和 **`phc2sys`** 程序，**`ptp4l`** 可实现 PTP 边界时钟和普通时钟。如果启用硬件时戳，**`ptp4l`** 会将 PTP 硬件时钟同步到主时钟。如果使用软件时戳，它会将系统时钟同步到主时钟。仅当使用硬件时戳将系统时钟同步到网络接口卡 (NIC) 上的 PTP 硬件时钟时，才需要 **`phc2sys`**。

20.2 使用 PTP

20.2.1 网络驱动程序和硬件支持

PTP 要求使用的内核网络驱动程序支持软件时戳或硬件时戳。此外，NIC 还必须支持物理硬件中的时戳。您可以使用 **ethtool** 校验驱动程序和 NIC 时戳功能：

```
> sudo ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

软件时戳需要以下参数：

```
SOF_TIMESTAMPING_SOFTWARE
SOF_TIMESTAMPING_TX_SOFTWARE
SOF_TIMESTAMPING_RX_SOFTWARE
```

硬件时戳需要以下参数：

```
SOF_TIMESTAMPING_RAW_HARDWARE
SOF_TIMESTAMPING_TX_HARDWARE
SOF_TIMESTAMPING_RX_HARDWARE
```

20.2.2 使用 ptp4l

ptp4l 默认使用硬件时戳。您需要以 root 身份使用 -i 选项指定支持硬件时戳的网络接口。-m 指示 **ptp4l** 将其输出列显到标准输出，而不是列显到系统的日志记录工具：

```
> sudo ptp4l -m -i eth0
selected eth0 as PTP clock
port 1: INITIALIZING to LISTENING on INITIALIZE
port 0: INITIALIZING to LISTENING on INITIALIZE
port 1: new foreign master 00a152.fffe.0b334d-1
selected best master clock 00a152.fffe.0b334d
port 1: LISTENING to UNCALIBRATED on RS_SLAVE
master offset -25937 s0 freq +0 path delay       12340
master offset -27887 s0 freq +0 path delay       14232
master offset -38802 s0 freq +0 path delay       13847
master offset -36205 s1 freq +0 path delay       10623
master offset  -6975 s2 freq -30575 path delay   10286
port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
master offset  -4284 s2 freq -30135 path delay    9892
```

master offset 值表示测得的与主时钟之间的偏差（以纳秒为单位）。

s0、s1 和 s2 指示器显示时钟伺服的不同状态：s0 表示已解锁，s1 表示时钟步进，s2 表示已锁定。如果伺服处于已锁定状态 (s2)，并且配置文件中的 pi_offset_const 选项设置为负值，则时钟不会步进，而只会缓慢调整（有关详细信息，请参见 **man 8 ptp4l**）。

freq 值表示时钟的频率调整（以十亿分率 (ppb) 为单位）。

path delay 值表示从主时钟发送的同步消息的预计延迟（以纳秒为单位）。

端口 0 是用于本地 PTP 管理的 Unix 域套接字。端口 1 是 eth0 接口。

比如，发生 INITIALIZE、RS_SLAVE 和 MASTER_CLOCK_SELECTED 事件时，端口状态就会变成 INITIALIZING、LISTENING、UNCALIBRATED 和 SLAVE。当端口状态从 UNCALIBRATED 更改为 SLAVE 时，表示计算机已与 PTP 主时钟成功同步。

您可以使用 -S 选项启用软件时戳。

```
> sudo ptp4l -m -S -i eth3
```

您还可以将 **ptp4l** 作为服务运行：

```
> sudo systemctl start ptp4l
```

在本例中，**ptp4l** 从 /etc/sysconfig/ptp4l 文件读取其选项。默认情况下，此文件指示 **ptp4l** 从 /etc/ptp4l.conf 读取配置选项。有关 **ptp4l** 选项和配置文件设置的详细信息，请参见 **man 8 ptp4l**。

要永久启用 **ptp4l** 服务，请运行以下命令：

```
> sudo systemctl enable ptp4l
```

要禁用该服务，请运行

```
> sudo systemctl disable ptp4l
```

20.2.3 ptp4l 配置文件

ptp4l 可以从可选的配置文件读取其配置。由于默认未使用任何配置文件，您需要通过 -f 指定配置文件。

```
> sudo ptp4l -f /etc/ptp4l.conf
```

配置文件分为若干部分。全局部分（通过 [global] 识别）用于设置程序选项、时钟选项和默认端口选项。其他部分与特定的端口相关，会覆盖默认端口选项。部分的名称是配置的端口的名称 — 例如 [eth0]。空白端口部分可用于替换命令行选项。

```
[global]
verbose               1
time_stamping         software
[eth0]
```

示例配置文件等效于以下命令选项：

```
> sudo ptp4l -i eth0 -m -S
```

有关完整的 **ptp4l** 配置选项列表，请参见 **man 8 ptp4l**。

20.2.4 延迟测量

ptp4l 通过两种方法测量时间延迟：**对等式** (P2P) 或**端到端** (E2E)。

P2P

此方法使用 `-P` 指定。

它可以更快地对网络环境中的更改做出反应，并可更准确地测量延迟。仅在每个端口都会与另一个端口交换 PTP 消息的网络中才会使用此方法。P2P 需受到通讯路径中所有硬件的支持。

E2E

此方法使用 `-E` 指定。此为默认设置。

自动选择方法

此方法使用 `-A` 指定。自动选项以 E2E 模式启动 **`ptp4l`**，如果收到了对等延迟请求，则会切换为 P2P 模式。

重要：常用测量方法

单个 PTP 通讯路径上的所有时钟必须使用相同的方法来测量时间延迟。如果在使用 E2E 机制的端口上收到了对等延迟请求，或者在使用 P2P 机制的端口上收到了 E2E 延迟请求，则会列显警告。

20.2.5 PTP 管理客户端：`pmc`

可以使用 **`pmc`** 客户端获取有关 **`ptp41`** 的更详细信息。pmc 从标准输入或命令行读取按名称和管理 ID 指定的操作。然后通过选定的传输方式发送操作，并列显收到的任何答复。pmc 支持以下三个操作：**`GET`** 可检索指定的信息，**`SET`** 可更新指定的信息，**`CMD`**（或 **`COMMAND`**）可发起指定的事件。

默认情况下，管理命令会在所有端口上寻址。可以使用 **`TARGET`** 命令为后续消息选择特定的时钟和端口。如需完整的管理 ID 列表，请运行 **`pmc help`**。

```
> sudo pmc -u -b 0 'GET TIME_STATUS_NP'
sending: GET TIME_STATUS_NP
        90f2ca.fffe.20d7e9-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
                master_offset              283
                ingress_time               1361569379345936841
                cumulativeScaledRateOffset   +1.000000000
                scaledLastGmPhaseChange    0
```

```
                gmTimeBaseIndicator        0
                lastGmPhaseChange          0x0000'0000000000000.0000
                gmPresent                  true
                gmIdentity                 00b058.feef.0b448a
```

`-b` 选项指定所发送消息中的边界跃点值。将此选项设置为 0 会将边界限制为本地 **`ptp4l`** 实例。如果将该值设置得更高，则还会检索距离本地实例更远的 PTP 节点发出的消息。返回的信息可能包括：

stepsRemoved

超级主时钟的通讯节点数。

offsetFromMaster、master_offset

该时钟与主时钟之间上次测得的偏差（纳秒）。

meanPathDelay

从主时钟发送的同步消息的预计延迟（纳秒）。

gmPresent

如果为 `true`，则表示 PTP 时钟已同步到主时钟；本地时钟不是超级主时钟。

gmIdentity

此为超级主时钟身份。

有关完整的 **`pmc`** 命令行选项列表，请参见 **`man 8 pmc`**。

20.3 使用 phc2sys 同步时钟

使用 **`phc2sys`** 可将系统时钟同步到网卡上的 PTP 硬件时钟 (PHC)。系统时钟被视为**从属时钟**，而网卡上的时钟则为**主时钟**。PHC 本身将与 **`ptp4l`** 同步（请参见第 20.2 节 “使用 PTP”）。使用 `-s` 可按设备或网络接口指定主时钟。使用 `-w` 等待 **`ptp4l`** 进入同步状态。

```
> sudo phc2sys -s eth0 -w
```

PTP 以**国际原子时** (TAI) 运行，而系统时钟使用的是**协调世界时** (UTC)。如果您不指定 `-w` 来等待 **`ptp4l`** 同步，可以使用 `-O` 来指定 TAI 与 UTC 之间的偏差（以秒为单位）：

```
> sudo phc2sys -s eth0 -O -35
```

也可以将 **phc2sys** 作为服务运行：

```
> sudo systemctl start phc2sys
```

在本例中，**phc2sys** 从 /etc/sysconfig/phc2sys 文件读取其选项。有关 **phc2sys** 选项的详细信息，请参见 **man 8 phc2sys**。

要永久启用 **phc2sys** 服务，请运行以下命令：

```
> sudo systemctl enable phc2sys
```

要禁用该服务，请运行

```
> sudo systemctl disable phc2sys
```

20.3.1 校验时间同步

当 PTP 时间同步正常工作并且使用了硬件时戳时，**ptp4l** 和 **phc2sys** 会定期向系统日志输出包含时间偏差和频率调节的消息。

ptp4l 输出示例：

```
ptp4l[351.358]: selected /dev/ptp0 as PTP clock
ptp4l[352.361]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[352.361]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[353.210]: port 1: new foreign master 00a069.eefe.0b442d-1
ptp4l[357.214]: selected best master clock 00a069.eefe.0b662d
ptp4l[357.214]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[359.224]: master offset       3304 s0 freq      +0 path delay      9202
ptp4l[360.224]: master offset       3708 s1 freq  -28492 path delay      9202
ptp4l[361.224]: master offset      -3145 s2 freq  -32637 path delay      9202
ptp4l[361.224]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[362.223]: master offset       -145 s2 freq  -30580 path delay      9202
ptp4l[363.223]: master offset       1043 s2 freq  -28436 path delay      8972
[...]
ptp4l[371.235]: master offset        285 s2 freq  -28511 path delay      9199
ptp4l[372.235]: master offset        -78 s2 freq  -28788 path delay      9204
```

phc2sys 输出示例：

```
phc2sys[616.617]: Waiting for ptp4l...
phc2sys[628.628]: phc offset     66341 s0 freq      +0 delay   2729
phc2sys[629.628]: phc offset     64668 s1 freq  -37690 delay   2726
[...]
phc2sys[646.630]: phc offset      -333 s2 freq  -37426 delay   2747
phc2sys[646.630]: phc offset       194 s2 freq  -36999 delay   2749
```

ptp4l 通常会频繁写入消息。可以使用 `summary_interval` 指令降低该频率。其值的表达式为 2 的 N 次幂。例如，要将输出频率降低为每隔 1024（等于 2^10）秒，请将下面一行添加到 `/etc/ptp4l.conf` 文件。

```
summary_interval 10
```

也可以使用 `-u SUMMARY-UPDATES` 选项降低 **phc2sys** 命令的更新频率。

20.4 配置示例

本节提供了几个 **ptp4l** 配置示例。这些示例不是完整的配置文件，而是要对特定文件所做更改的精简列表。字符串 `ethX` 表示设置中的实际网络接口名称。

例 20.1： 使用软件时戳的从属时钟

> `/etc/sysconfig/ptp4l`:
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> 未对分发包 `/etc/ptp4l.conf` 进行任何更改。

例 20.2： 使用硬件时戳的从属时钟

> `/etc/sysconfig/ptp4l`:
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> `/etc/sysconfig/phc2sys`:
>
> ```
> OPTIONS="-s ethX -w"
> ```
>
> 未对分发包 `/etc/ptp4l.conf` 进行任何更改。

例 20.3： 使用硬件时戳的主时钟

> /etc/sysconfig/ptp4l:
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> /etc/sysconfig/phc2sys:
>
> ```
> OPTIONS="-s CLOCK_REALTIME -c ethX -w"
> ```
>
> /etc/ptp4l.conf:
>
> ```
> priority1 127
> ```

例 20.4： 使用软件时戳的主时钟（一般不建议使用）

> /etc/sysconfig/ptp4l:
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> /etc/ptp4l.conf:
>
> ```
> priority1 127
> ```

20.5 PTP 和 NTP

NTP 和 PTP 时间同步工具可以共存，可实现双向时间同步。

20.5.1 NTP 到 PTP 的同步

使用 chronyd 同步本地系统时钟时，可将 **ptp4l** 配置为超级主时钟，以便通过 PTP 从本地系统时钟分发时间。在 /etc/ptp4l.conf 中包含 priority1 选项：

```
[global]
priority1 127
[eth0]
```

然后运行 **ptp4l**：

```
> sudo ptp4l -f /etc/ptp4l.conf
```

使用硬件时戳时，需要通过 **phc2sys** 将 PTP 硬件时钟同步到系统时钟：

```
> sudo phc2sys -c eth0 -s CLOCK_REALTIME -w
```

20.5.2 配置 PTP-NTP 网桥

在未配备支持 PTP 的交换机或路由器的网络中，如果可以使用高度精确的 PTP 超级主时钟，则计算机可以作为 PTP 从属和第 1 层 NTP 服务器运行。此类计算机需有两个或更多网络接口，并且靠近或者能够直接连接到超级主时钟。这可以确保在网络中实现高度精确的同步。

将 **ptp4l** 和 **phc2sys** 程序配置为使用一个网络接口来通过 PTP 同步系统时钟。然后将 `chronyd` 配置为使用另一接口提供系统时间：

```
bindaddress 192.0.131.47
hwtimestamp eth1
local stratum 1
```

注意：NTP 和 DHCP

默认情况下，当 DHCP 客户端命令 **dhclient** 收到 NTP 服务器列表时，会将这些服务器添加到 NTP 配置。为防止出现此行为，请设置

```
NETCONFIG_NTP_POLICY=""
```

（在 `/etc/sysconfig/network/config` 文件中）。

A GNU licenses

This appendix contains the GNU Free Documentation License version 1.2.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/copyleft/ ↗.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled “GNU
Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.