

Краткое содержание

Благодарности	23
Предисловие	24
Глава 1. Общая картина	28
Глава 2. Основные команды и иерархия каталогов	39
Глава 3. Устройства	79
Глава 4. Диски и файловые системы	102
Глава 5. Загрузка ядра Linux	156
Глава 6. Запуск пользовательского пространства	176
Глава 7. Настройка системы: журналирование, системное время, пакетные задачи и пользователи	208
Глава 8. Процессы и использование ресурсов	244
Глава 9. Сеть и ее конфигурация	270
Глава 10. Сетевые приложения и службы	320
Глава 11. Сценарии оболочки	344
Глава 12. Передача файлов по сети и доступ к ним	368
Глава 13. Пользовательское окружение	390
Глава 14. Краткий обзор рабочего стола Linux. Вывод на печать	402
Глава 15. Инструменты разработки	419
Глава 16. Компиляция программ из исходного кода на языке C	442
Глава 17. Виртуализация	458
Библиография	478

Оглавление

Об авторе	22
О научных редакторах	22
Благодарности	23
Предисловие	24
Для кого эта книга	24
Требуемый уровень знаний	24
Как читать книгу	25
Практический подход	25
Как устроена эта книга	25
Новое в третьем издании	26
Немного о терминологии	27
От издательства	27
Глава 1. Общая картина	28
1.1. Уровни абстракции в системе Linux	29
1.2. Оборудование: оперативная память	31
1.3. Ядро	32
1.3.1. Управление процессами	32
1.3.2. Управление памятью	34
1.3.3. Управления драйверами устройств	34
1.3.4. Системные вызовы и поддержка	35
1.4. Пользовательское пространство	36
1.5. Пользователи	37
1.6. Что дальше?	38
Глава 2. Основные команды и иерархия каталогов	39
2.1. Оболочка Bourne Shell (bash): /bin/sh	40
2.2. Использование оболочки	40

2.2.1. Окно оболочки	40
2.2.2. Программа cat	41
2.2.3. Стандартный поток ввода (stdin) и стандартный поток вывода (stdout)	42
2.3. Основные команды	43
2.3.1. Команда ls	43
2.3.2. Команда cp	44
2.3.3. Команда mv	44
2.3.4. Команда touch	44
2.3.5. Команда rm	44
2.3.6. Команда echo	45
2.4. Перемещение по каталогам	45
2.4.1. Команда cd	45
2.4.2. Команда mkdir	46
2.4.3. Команда rmdir	46
2.4.4. Шаблоны поиска (переменные Wildcards)	46
2.5. Команды среднего уровня	47
2.5.1. Команда grep	48
2.5.2. Команда less	48
2.5.3. Команда pwd	49
2.5.4. Команда diff	49
2.5.5. Команда file	50
2.5.6. Команды find и locate	50
2.5.7. Команды head и tail	50
2.5.8. Команда sort	51
2.6. Смена пароля и оболочки	51
2.7. Файлы с точками	51
2.8. Переменные окружения и оболочки	52
2.9. Переменная пути PATH	52
2.10. Специальные символы	53
2.11. Редактирование в командной строке	54
2.12. Текстовые редакторы	55
2.13. Онлайн-поддержка	56
2.14. Ввод и вывод командной оболочки	58
2.14.1. Стандартная ошибка	59
2.14.2. Стандартное перенаправление ввода	59

2.15. Сообщения об ошибках	60
2.15.1. Структура сообщений об ошибках в Unix	60
2.15.2. Распространенные ошибки	61
2.16. Перечисление процессов и управление ими	62
2.16.1. Параметры команды ps	63
2.16.2. Завершение процесса	64
2.16.3. Управление заданиями	65
2.16.4. Фоновые процессы	65
2.17. Режимы файлов и права доступа	66
2.17.1. Изменение прав доступа	68
2.17.2. Использование символических ссылок	69
2.18. Архивирование и сжатие файлов	71
2.18.1. Утилита gzip	71
2.18.2. Утилита tar	71
2.18.3. Сжатые архивы (.tar.gz)	72
2.18.4. Утилита zcat	73
2.18.5. Другие утилиты сжатия	73
2.19. Основная иерархия каталогов Linux	74
2.19.1. Другие корневые подкаталоги	76
2.19.2. Каталог /usr	76
2.19.3. Местонахождение ядра	76
2.20. Запуск команд от имени суперпользователя	77
2.20.1. Команда sudo	77
2.20.2. Файл /etc/sudoers	77
2.20.3. Журналы sudo	78
2.21. Что дальше?	78
Глава 3. Устройства	79
3.1. Файлы устройств	79
3.2. Путь к устройству sysfs	81
3.3. Утилита dd и устройства	82
3.4. Имена устройств	83
3.4.1. Жесткие диски: /dev/sd*	84
3.4.2. Виртуальные диски: /dev/xvd*, /dev/vd*	85
3.4.3. Устройства долговременной памяти: /dev/nvme*	85

3.4.4. Подсистема создания виртуальных блочных устройств: /dev/dm-*, /dev/mapper/*	85
3.4.5. CD- и DVD-приводы: /dev/sr*	85
3.4.6. Жесткие диски PATA: /dev/hd*	86
3.4.7. Терминалы: /dev/tty*, /dev/pts/* и /dev/tty	86
3.4.8. Последовательные порты: /dev/ttyS*, /dev/ttyUSB*, /dev/ttyACM*	87
3.4.9. Параллельные порты: /dev/lp0 и /dev/lp1	88
3.4.10. Аудиоустройства: /dev/snd/*, /dev/dsp, /dev/audio и другие	88
3.4.11. Создание файла устройства	88
3.5. Менеджер устройств udev	89
3.5.1. Файловая система devtmpfs	90
3.5.2. Работа и настройка менеджера udevd	91
3.5.3. Команда udevadm	93
3.5.4. Отслеживание устройств	94
3.6. Подробнее об интерфейсе SCSI и ядре Linux	95
3.6.1. USB-накопитель и SCSI	98
3.6.2. Интерфейсы SCSI и ATA	99
3.6.3. Обобщенные устройства SCSI	99
3.6.4. Методы множественного доступа к одному устройству	100

Глава 4. Диски и файловые системы

4.1. Разбиение дисков на разделы	104
4.1.1. Просмотр таблицы разделов	105
4.1.2. Редактирование таблиц разделов	108
4.1.3. Создание таблицы разделов	109
4.1.4. Геометрия дисков и разделов	111
4.1.5. Чтение твердотельных дисков	113
4.2. Файловые системы	114
4.2.1. Типы файловых систем	114
4.2.2. Создание файловой системы	116
4.2.3. Монтирование файловой системы	117
4.2.4. Идентификатор UUID файловой системы	119
4.2.5. Буферизация диска, кэширование и файловые системы	120
4.2.6. Параметры монтирования файловой системы	120

4.2.7. Повторное монтирование файловой системы	122
4.2.8. Таблица файловой системы /etc/fstab	122
4.2.9. Альтернативы файлу /etc/fstab	124
4.2.10. Емкость файловой системы	124
4.2.11. Проверка и восстановление файловых систем	126
4.2.12. Файловые системы специального назначения	129
4.3. Область подкачки swap	130
4.3.1. Использование раздела диска в качестве области подкачки ..	130
4.3.2. Использование файла в качестве области подкачки	131
4.3.3. Определение необходимого размера области подкачки	131
4.4. Менеджер логических томов LVM	132
4.4.1. Работа с менеджером LVM	134
4.4.2. Реализация менеджера LVM	145
4.5. Что дальше? Диски и пользовательское пространство	149
4.6. Что находится внутри традиционной файловой системы	150
4.6.1. Сведения о дескрипторе и количество ссылок	152
4.6.2. Распределение блоков	154
4.6.3. Работа с файловыми системами в пользовательском пространстве	154
Глава 5. Загрузка ядра Linux	156
5.1. Сообщения при загрузке	157
5.2. Параметры инициализации и загрузки ядра	158
5.3. Параметры ядра	159
5.4. Загрузчики	160
5.4.1. Задачи загрузчика	161
5.4.2. Обзор загрузчиков	162
5.5. Введение в загрузчик GRUB	163
5.5.1. Изучение устройств и разделов с помощью командной строки GRUB	165
5.5.2. Конфигурация GRUB	167
5.5.3. Установка GRUB	170
5.6. Проблемы безопасной загрузки UEFI	171
5.7. Метод цепной загрузки других операционных систем	172
5.8. Детали работы загрузчика	173
5.8.1. Загрузчик MBR	173

5.8.2. Загрузчик UEFI	173
5.8.3. Как работает GRUB	174
Глава 6. Запуск пользовательского пространства	176
6.1. Основные сведения об init	177
6.2. Определение системы инициализации	178
6.3. systemd	178
6.3.1. Юниты и типы юнитов	179
6.3.2. Графики загрузки и зависимостей юнитов	179
6.3.3. Конфигурация systemd	180
6.3.4. Процесс работы systemd	183
6.3.6. Зависимости systemd	188
6.3.7. Запуск по запросу и параллелизация ресурсов в systemd	191
6.3.8. Вспомогательные компоненты systemd	196
6.4. Уровни выполнения в System V	197
6.5. System V init	198
6.5.1. System V init: последовательность команд при запуске	199
6.5.2. Ферма ссылок System V init	200
6.5.3. Команда run-parts	202
6.5.4. Управление System V init	202
6.5.5. Совместимость systemd и System V	203
6.6. Завершение работы системы	203
6.7. Начальная файловая система оперативной памяти	205
6.8. Аварийная загрузка системы и однопользовательский режим	206
6.9. Что дальше?	207
Глава 7. Настройка системы: журналирование, системное время, пакетные задачи и пользователи	208
7.1. Ведение системного журнала	209
7.1.1. Проверка настроек журнала	209
7.1.2. Поиск и мониторинг журналов	210
7.1.3. Ротация файлов журнала	214
7.1.4. Обслуживание журналов journald	215
7.1.5. Детали системного журналирования	215
7.2. Структура каталога /etc	218
7.3. Файлы управления пользователями	218
7.3.1. Файл /etc/passwd	219

7.3.2. Особые пользователи	220
7.3.3. Файл /etc/shadow	221
7.3.4. Управление пользователями и паролями	221
7.3.5. Работа с группами пользователей	222
7.4. Команды <code>getty</code> и <code>login</code>	223
7.5. Установка времени	224
7.5.1. Представление времени ядра и часовые пояса	224
7.5.2. Сетевое время	225
7.6. Планирование повторяющихся задач с помощью команды <code>cron</code> и юнитов таймера	226
7.6.1. Установка файлов <code>Crontab</code>	227
7.6.2. Системные файлы <code>Crontab</code>	227
7.6.3. Юниты таймера	228
7.6.4. Утилита <code>cron</code> против юнитов таймера	230
7.7. Планирование разовых задач с помощью службы <code>at</code>	230
7.7.1. Аналоги таймера	231
7.8. Юниты таймера обычных пользователей	231
7.9. Доступ пользователя	232
7.9.1. ID пользователей и переключение пользователей	232
7.9.2. Владельцы процессов, действующий UID, реальный UID и сохраненный UID	233
7.9.3. Идентификация пользователя, аутентификация и авторизация	235
7.9.4. Применение библиотек для получения информации о пользователе	236
7.10. Подключаемые модули аутентификации (PAM)	237
7.10.1. Конфигурация PAM	237
7.10.2. Советы по синтаксису конфигурации PAM	241
7.10.3. Модули PAM и пароли	242
7.11. Что дальше?	243
Глава 8. Процессы и использование ресурсов	244
8.1. Отслеживание процессов	244
8.2. Поиск открытых файлов с помощью команды <code>lsdf</code>	245
8.2.1. Считывание вывода команды <code>lsdf</code>	245
8.2.2. Использование команды <code>lsdf</code>	247

8.3. Отслеживание выполнения программ и системных вызовов	247
8.3.1. Команда <code>strace</code>	247
8.3.2. Команда <code>ltrace</code>	249
8.4. Потоки	250
8.4.1. Однопоточные и многопоточные процессы	250
8.4.2. Просмотр потоков	250
8.5. Введение в мониторинг ресурсов	252
8.5.1. Измерение процессорного времени	252
8.5.2. Настройка приоритетов процесса	253
8.5.3. Измерение производительности процессора с помощью среднего значения загрузки	254
8.5.4. Мониторинг состояния памяти	255
8.5.5. Мониторинг производительности процессора и памяти с помощью команды <code>vmstat</code>	258
8.5.6. Мониторинг ввода-вывода I/O	260
8.5.7. Мониторинг каждого процесса с помощью команды <code>pidstat</code> ..	262
8.6. Группы управления (<code>cgroups</code>)	263
8.6.1. Различие между версиями <code>cgroup</code>	264
8.6.2. Просмотр <code>cgroups</code>	266
8.6.3. Управление и создание <code>cgroups</code>	267
8.6.4. Отображение использования ресурсов	268
8.7. Дополнительно	269
Глава 9. Сеть и ее конфигурация	270
9.1. Основы сети	271
9.2. Пакеты	271
9.3. Сетевые уровни	272
9.4. Сетевой уровень	273
9.4.1. Просмотр IP-адреса	275
9.4.2. Подсети	275
9.4.3. Распространенные маски подсети и нотация CIDR	276
9.5. Маршруты и таблица маршрутизации ядра	277
9.6. Шлюз по умолчанию	278
9.7. IPv6-адреса и сети	279
9.7.1. Просмотр конфигурации IPv6 в системе	280
9.7.2. Настройка сетей с двумя стеками	281

9.8. Основные инструменты ICMP и DNS	281
9.8.1. Команда ping	282
9.8.2. Служба DNS и команда host	283
9.9. Физический уровень и сеть Ethernet	283
9.10. Сетевые интерфейсы ядра	284
9.11. Введение в настройки сетевого интерфейса	285
9.11.1. Ручная настройка интерфейсов	285
9.11.2. Добавление и удаление маршрутов вручную	286
9.12. Конфигурация сети, активируемая при загрузке	287
9.13. Проблемы с настройкой сети вручную и при загрузке	288
9.14. Менеджеры настройки сети	289
9.14.1. Работа NetworkManager	289
9.14.2. Взаимодействие с NetworkManager	290
9.14.3. Настройка NetworkManager	290
9.15. Разрешения сетевых имен	292
9.15.1. Файл /etc/hosts	293
9.15.2. Файл resolv.conf	293
9.15.3. Кэширование и DNS с нулевой конфигурацией	294
9.15.4. Файл /etc/nsswitch.conf	295
9.16. Localhost	296
9.17. Транспортный уровень: TCP, UDP и службы	296
9.17.1. TCP-порты и соединения	297
9.17.2. Протокол UDP	301
9.18. Возврат к простой локальной сети	301
9.19. Протокол DHCP	302
9.19.1. DHCP-клиенты Linux	302
9.19.2. DHCP-серверы Linux	303
9.20. Автоматическая настройка сети IPv6	303
9.21. Настройка Linux в качестве маршрутизатора	304
9.22. Частные сети (IPv4)	306
9.23. Преобразование сетевых адресов NAT (маскарадинг IP-адресов) ..	306
9.24. Linux и маршрутизаторы	308
9.25. Брандмауэры	309
9.25.1. Основы брандмауэров Linux	310
9.25.2. Настройка правил брандмауэра	311
9.25.3. Стратегии брандмауэра	313

9.26. Ethernet, IP, ARP и NDP	315
9.27. Беспроводная сеть Ethernet	317
9.27.1. Утилита iw	318
9.27.2. Безопасность беспроводной сети	318
9.28. Выводы	319
Глава 10. Сетевые приложения и службы	320
10.1. Основы работы служб	320
10.2. Взглянем глубже	322
10.3. Сетевые серверы	323
10.3.1. Служба защищенной оболочки SSH	324
10.3.2. Сервер sshd	326
10.3.3. Утилита fail2ban	328
10.3.4. SSH-клиент	329
10.4. Сетевые серверы до systemd: inetd/xinetd	330
10.5. Средства диагностики	331
10.5.1. Утилита lsof	332
10.5.2. Утилита tcpdump	333
10.5.3. Утилита netcat	335
10.5.4. Сканирование портов	336
10.6. Удаленный вызов процедур	337
10.7. Безопасность сети	338
10.7.1. Типичные уязвимости	339
10.7.2. Ресурсы безопасности	340
10.8. Что дальше?	341
10.9. Сетевые сокеты	341
10.10. Доменные сокеты Unix	342
Глава 11. Сценарии оболочки	344
11.1. Основы сценариев оболочки	344
11.1.1. Ограничения скриптов оболочки	345
11.2. Кавычки и литералы	346
11.2.1. Литералы	346
11.2.2. Одинарные кавычки	347
11.2.3. Двойные кавычки	348
11.2.4. Литерал с одинарными кавычками	348

11.3. Специальные переменные	349
11.3.1. Индивидуальные аргументы: \$1, \$2 и другие.....	349
11.3.2. Количество аргументов: \$#.....	350
11.3.3. Все аргументы: \$@	350
11.3.4. Имя сценария: \$0	350
11.3.5. ID процесса: \$\$	351
11.3.6. Код возврата: \$?	351
11.4. Коды возврата	351
11.5. Условные операторы	352
11.5.1. Обходной путь для предотвращения ошибки Empty Parameter Lists	353
11.5.2. Другие команды для проверки условий	353
11.5.3. Ключевое слово elif	353
11.5.4. Логические конструкции	354
11.5.5. Проверка условий	355
11.5.6. Ключевое слово case	357
11.6. Циклы	358
11.6.1. Циклы for	358
11.6.2. Циклы while	359
11.7. Подстановка команд	359
11.8. Управление временными файлами	360
11.9. Неге-документы	362
11.10. Важные утилиты сценариев оболочки	362
11.10.1. Утилита basename	362
11.10.2. Утилита awk	363
11.10.3. Утилита sed	363
11.10.4. Утилита xargs	364
11.10.5. Утилита expr	365
11.10.6. Утилита exec	365
11.11. Подоболочки	366
11.12. Добавление файлов в скрипты	366
11.13. Чтение пользовательского ввода	367
11.14. Когда не стоит использовать сценарии оболочки	367
Глава 12. Передача файлов по сети и доступ к ним	368
12.1. Быстрое копирование данных	369

12.2. Утилита rsync	370
12.2.1. Начало работы с rsync	370
12.2.2. Создание точной копии структуры каталогов	371
12.2.3. Добавление косой черты	372
12.2.4. Исключение файлов и каталогов	373
12.2.5. Проверка копирования, меры предосторожности и подробный режим	374
12.2.6. Сжатие данных	375
12.2.7. Ограничение ширины полосы пропускания	375
12.2.8. Передача файлов на ваш компьютер	376
12.2.9. Больше информации о rsync	376
12.3. Совместное использование файлов	376
12.3.1. Производительность совместного использования файлов ...	377
12.3.2. Безопасность совместного доступа к файлам	378
12.4. Совместное использование файлов с помощью Samba	378
12.4.1. Настройка сервера	379
12.4.2. Контроль доступа к серверу	379
12.4.3. Пароли	380
12.4.4. Запуск сервера вручную	382
12.4.5. Диагностика и файлы журналов	382
12.4.6. Настройка совместного использования файлов	382
12.4.7. Домашние каталоги	383
12.4.8. Совместное применение принтеров	383
12.4.9. Клиенты сервера Samba	384
12.5. Клиентская программа SSHFS	386
12.6. NFS	387
12.7. Облачное хранилище	388
12.8. Состояние совместного доступа к файлам в сети	388
Глава 13. Пользовательское окружение	390
13.1. Создание файлов запуска	391
13.2. Изменение файлов запуска	391
13.3. Элементы файла запуска оболочки	391
13.3.1. Путь к команде	392
13.3.2. Путь к странице руководства	393
13.3.3. Приглашения prompt	393

13.3.4. Псевдонимы	394
13.3.5. Маска прав доступа	394
13.4. Порядок и примеры файлов запуска	395
13.4.1. Оболочка bash	395
13.4.2. Оболочка tcsh	398
13.5. Пользовательские настройки по умолчанию	399
13.5.1. Параметры оболочки по умолчанию	399
13.5.2. Текстовый редактор	400
13.5.3. Пейджер	400
13.6. Подводные камни в файлах запуска	400
13.7. Далее о файлах запуска	401
Глава 14. Краткий обзор рабочего стола Linux. Вывод на печать	402
14.1. Компоненты рабочего стола	403
14.1.1. Фреймбуфер	403
14.1.2. Система X Window	403
14.1.3. Протокол Wayland	404
14.1.4. Менеджеры окон	404
14.1.5. Наборы инструментов	405
14.1.6. Окружение рабочего стола	405
14.1.7. Приложения	405
14.2. Wayland или X?	406
14.3. Обзор протокола Wayland	406
14.3.1. Композитный менеджер окон	406
14.3.2. Библиотека libinput	407
14.3.3. Совместимость X и Wayland	408
14.4. Обзор системы X Window	409
14.4.1. Дисплейные менеджеры	410
14.4.2. Сетевая прозрачность	411
14.4.3. Способы исследования X-клиентов	411
14.4.4. События на сервере (X events)	412
14.4.5. X-ввод и настройки предпочтений	413
14.5. Шина D-Bus	415
14.5.1. Системные и сеансовые экземпляры	416
14.5.2. Мониторинг сообщений D-Bus	416

14.6. Печать	417
14.6.1. Сервер печати CUPS	417
14.6.2. Фильтры преобразования форматов и печати	418
14.7. Дополнительно об окружениях рабочего стола	418
Глава 15. Инструменты разработки	419
15.1. Компилятор C	419
15.1.1. Компиляция нескольких исходных файлов	420
15.1.2. Связывание с библиотеками	422
15.1.3. Разделяемые библиотеки	423
15.1.4. Файлы заголовков (Include) и каталоги	428
15.2. Утилита make	430
15.2.1. Makefile	431
15.2.2. Встроенные правила	432
15.2.3. Финальная сборка программы	432
15.2.4. Обновление зависимостей	433
15.2.5. Аргументы и параметры командной строки	433
15.2.6. Стандартные макросы и переменные	434
15.2.7. Стандартные цели	435
15.2.8. Организация файла Makefile	436
15.3. Программы Lex и Yacc	437
15.4. Языки сценариев	437
15.4.1. Язык Python	438
15.4.2. Язык Perl	439
15.4.3. Другие языки сценариев	439
15.5. Язык Java	440
15.6. А что дальше? Компиляция пакетов	441
Глава 16. Компиляция программ из исходного кода на языке C	442
16.1. Системы сборки программ	443
16.2. Распаковка пакетов с исходным кодом C	443
16.3. Утилита GNU Autoconf	445
16.3.1. Пример использования Autoconf	446
16.3.2. Установка с помощью инструментов создания пакетов	447
16.3.3. Параметры сценария configure	447
16.3.4. Переменные окружения	448

16.3.5. Цели Autoconf	449
16.3.6. Файлы журналов Autoconf	450
16.3.7. Утилита pkg-config	450
16.4. Процесс установки	452
16.4.1. Места установки	453
16.5. Применение исправлений	453
16.6. Устранение ошибок компиляции и установки	454
16.6.1. Особые ошибки	455
16.7. Что дальше?	457
Глава 17. Виртуализация	458
17.1. Виртуальные машины	458
17.1.1. Гипервизоры	459
17.1.2. Оборудование виртуальной машины	460
17.1.3. Использование виртуальных машин	461
17.1.4. Недостатки виртуальных машин	462
17.2. Контейнеры	463
17.2.1. Docker, Podman и привилегии	464
17.2.2. Пример использования инструмента Docker	465
17.2.3. Система LXC	473
17.2.4. Платформа Kubernetes	473
17.2.5. Ошибки работы контейнеров	474
17.3. Виртуализация времени исполнения	476
Библиография	478

Если вы интересуетесь системой Linux, книга «Внутреннее устройство Linux» подойдет вам лучше всего.

LinuxInsider

В книге множество информации практически по всем аспектам архитектуры Linux.

Everyday Linux User

Вы получите полноценное представление о том, что происходит внутри системы, не увязая во множестве деталей. Это отличное дополнение к литературе по Linux.

*Фил Булл, соавтор книги Ubuntu Made Easy и член команды
по составлению документации для Ubuntu*

Автор погружается в глубину операционных систем на базе Linux и рассказывает, как все части системы сочетаются друг с другом.

DistroWatch

Книга заслуживает места на полке суперпользователя Linux.

Журнал The MagPi

Об авторе

Брайан Уорд работает с операционной системой Linux с 1993 года. Кроме этой, он написал книги *The Linux Kernel HOWTO*, *The Book of VMware* и *The Linux Problem Solver*.

О научных редакторах

Жорди Гутьеррес Эрмосо — профессиональный разработчик и пользователь GNU/Linux с почти двадцатилетним опытом работы, внесший вклад в развитие GNU Octave и Mercurial. В разное время он работал с криптографией, медицинской визуализацией и в сфере экологии — везде на Linux. Когда Жорди не сидит за компьютером, то занимается плаванием, математикой и вязанием.

Петрос Кутупис в настоящее время старший инженер по производительности программного обеспечения в компании HPE (ранее Cray Inc.) в подразделении Lustre High Performance File System. Он создатель RapidDisk Project (www.rapiddisk.org) и занимается его сопровождением. Петрос более десяти лет работает в индустрии хранения данных и помог внедрить многие современные технологии.

Благодарности

Вклад в эту книгу внесли не только те, кто участвовал в процессе ее создания, но и те, без кого я бы ничего не узнал о Linux, а именно: Джеймс Дункан, Дуглас Н. Арнольд, Билл Феннер, Кен Хорнстайн, Скотт Диксон, Дэн Эрлих, Феликс Ли и Грегори П. Смит. За помощь в работе над предыдущими изданиями благодарю Кароля Хурадо, Лорел Чун, Серену Янг, Элисон Лоу, Райли Хоффмана, Скотта Шварца, Дэна Салли, Доминика Пулена, Дональда Кейрона и Джину Стил.

В подготовке третьего издания участвовали Барбара Куин, Рэйчел Монаган, Джилл Франклин, Ларри Уэйда, Хорди Гутьерреса Эрмосо и Петрос Кутуписа. Билл Поллок, основатель издательства No Starch Press, сыграл особую роль в создании этой книги с ее первого издания. И еще раз благодарю Синьцзю Сие за то, что вытерпел меня и в этот раз.

Предисловие

Операционная система не должна быть загадкой. Любому специалисту хочется использовать свое программное обеспечение по желанию, без магических заклинаний или ритуалов. Чтобы этого добиться, необходимо понимать, что делает программное обеспечение и как оно работает, и именно этому посвящена данная книга. Больше вам никогда не придется сражаться со своим компьютером.

Linux — отличная платформа для обучения, так как она ничего не пытается скрыть от пользователя. Большинство сведений о конфигурации системы можно найти в легкочитаемых текстовых файлах. Единственная сложность — выяснить, какие части за что отвечают и как они все сочетаются друг с другом.

Для кого эта книга

Интерес к устройству Linux затрагивает разные сферы жизни. Профессионалы в сфере DevOps и разработчики должны знать почти всю информацию, которая рассматривается в этой книге. Архитекторы и разработчики программного обеспечения Linux также должны знать это, чтобы пользоваться операционной системой наилучшим образом. Для исследователей и студентов, часто работающих в своих собственных системах Linux, будет полезно узнать, почему в системе все устроено именно так, а не иначе, что и рассказывается в книге. Кроме того, есть еще и любители — люди, которые просто проводят время за своими компьютерами ради развлечения, выгоды или и того и другого сразу.

Хотите знать, почему одни вещи работают, а другие нет? Вам интересно, что произойдет, если что-либо изменить? Если вы ответили «Да!», то вы, скорее всего, любитель и найдете ответы на свои вопросы в этой книге.

Требуемый уровень знаний

Вам необязательно быть программистом, чтобы прочитать эту книгу. Понадобятся только базовые навыки пользователя компьютера: вы должны ориентироваться в графическом интерфейсе (особенно в интерфейсе установки и настроек для дистрибутива Linux) и знать, что такое файлы и каталоги (папки). Кроме того, будьте готовы искать и проверять дополнительную документацию в вашей системе и в интернете. И самое главное — быть готовыми исследовать свой компьютер.

Как читать книгу

Когда речь идет о технических предметах, донести все необходимые знания, — непростая задача. С одной стороны, читатель увязает в излишних подробностях и с трудом усваивает суть, поскольку человеческий разум просто не может одновременно обработать большое количество новых понятий. С другой — отсутствие подробностей приводит к тому, что читатель получает лишь смутное представление о предмете и не готов к усвоению дальнейшего материала.

В этой книге я упростил изложение и структурировал материал. В большинстве глав важная информация, которая необходима для дальнейшей работы, предлагается в первую очередь. По мере чтения главы вы встретите в ней и дополнительный материал. Надо ли вам сразу усваивать эти частности? В большинстве случаев полагаю, что нет. Если ваши глаза начинают тускнеть при виде большого количества подробностей, относящихся к только что изученному материалу, не раздумывая переходите к следующей главе или сделайте перерыв. Вас ожидают другие важные вещи.

Практический подход

Для работы вам понадобится компьютер с операционной системой Linux. Возможно, вы предпочтете виртуальную установку — для проверки большей части материала данной книги я пользовался приложением VirtualBox. Вы должны обладать правами доступа `superuser` (`root`), хотя основную часть времени следует использовать учетную запись обычного пользователя. Вы будете работать главным образом в командной строке, окне терминала или в удаленном сеансе. Если вы нечасто работали в этой среде, ничего страшного, в главе 2 вы узнаете об этом подробнее.

Как правило, команды будут выглядеть следующим образом:

```
$ ls /  
[some output]
```

Вводите текст, который выделен жирным шрифтом; обычным шрифтом показан ответный текст, который выдаст машина. Символ `$` является приглашением для пользователя с обычной учетной записью. Если вы увидите в качестве приглашения символ `#`, следует работать в учетной записи `superuser` (подробнее об этом в главе 2).

Как устроена эта книга

Я сгруппировал главы книги в три основные части. Первая часть — вводная, дает представление о системе в целом, а затем предлагается ряд практических заданий с инструментами, которые понадобятся вам для дальнейшей работы в системе. Далее вы детально изучите каждую часть системы, начиная с управления оборудованием и заканчивая конфигурацией сети, следуя обычному порядку запуска системы.

И наконец, вы познакомитесь с частями работающей системы, освоите необходимые навыки и рассмотрите инструменты, которые используют программисты.

В большинстве первых глав (кроме главы 2) активно задействовано ядро системы Linux, но по мере продвижения по книге вы будете работать и в своем пространстве пользователя. Если вы не понимаете, о чем я сейчас говорю, не беспокойтесь, объяснения будут даны в главе 1.

Материал излагается по возможности без привязки к какому-либо дистрибутиву системы. Было бы скучно описывать все варианты системы, поэтому я попытался рассказать о двух основных семействах дистрибутивов: Debian (включая Ubuntu) и RHEL/Fedora/CentOS. Кроме того, я описал настольные и серверные установки. Значительный объем материала можно использовать во встроенных системах, таких как Android и OpenWRT, но поиск различий между ними предоставляется вам.

Новое в третьем издании

Второе издание вышло в переходный период для всех систем Linux. Часть традиционных компонентов постепенно изменялась, что затрудняло изучение, потому что читатели сталкивались с большим разнообразием настроек. Однако сейчас новые элементы (в частности, `systemd`) надежно заняли свои места в системах, поэтому я смог значительно упростить и сократить материал по этой теме.

Я все так же сохранил акцент на роли ядра в системе Linux. Именно эта информация была наиболее интересна читателям, и вы сами, скорее всего, взаимодействуете с ядром чаще, чем думаете.

В новом издании появилась глава о виртуализации. Несмотря на то что Linux часто устанавливают именно через виртуальные машины (например, через облачные серверы), подобный способ виртуализации выходит за рамки данной книги. Все потому, что система работает на виртуальной машине практически так же, как на «голом» железе. Таким образом, информация об этом отличается в основном лишь терминологией. Кроме того, со времени выхода второго издания контейнеризация стала более популярной, поэтому я добавил информацию и о ней, ведь контейнеры в основном состоят из множества функций Linux, которые описаны в книге. В контейнерах часто используются контрольные группы `cgroups`, о которых я также рассказываю в третьем издании. В книгу в том числе добавлены сведения о менеджере логических томов, системе ведения журнала `journald` и протоколе IPv6 (в главе 9).

Мне хотелось снабдить вас сведениями, которые понадобятся для быстрого начала работы. Их усвоение потребует некоторых усилий, однако я не намереваюсь делать из вас «тяжелоатлетов», чтобы вы смогли одолеть эту книгу. Когда вы будете понимать важнейшие моменты, изложенные здесь, для вас не составит труда отыскать подробности и разобраться в них.

Я удалил кое-какие исторические детали, которые были в первом издании. Если вы интересуетесь системой Linux и ее отношением к истории системы Unix, обратитесь к книге Питера Салуса *The Daemon, the Gnu, and the Penguin* (Reed Media Services, 2008) — в ней рассказано о том, как развивалось используемое нами программное обеспечение.

Немного о терминологии

До сих пор ведутся споры о наименованиях некоторых элементов операционных систем. Они затрагивают даже само название системы Linux — как она должна называться: Linux или же GNU/Linux (чтобы отразить применение некоторых элементов проекта GNU)? В книге я старался использовать самые распространенные и по возможности наименее громоздкие названия.

От издательства

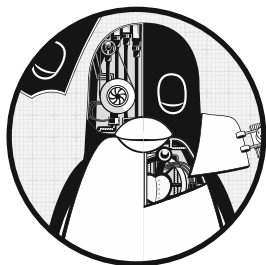
Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1

Общая картина



На первый взгляд такая современная операционная система, как Linux, очень сложна и состоит из невероятного количества инструментов, которые работают и взаимодействуют друг с другом одновременно. Например, веб-сервер может взаимодействовать с сервером базы данных, который, в свою очередь, может применять разделяемую библиотеку, используемую многими другими программами. Как же все это работает и какой во всем этом смысл?

Самый эффективный способ понять, как работает операционная система, — абстрагироваться от большинства деталей, составляющих часть, которую вы изучаете, и сосредоточиться на ее основной цели и функциональности. Например, когда вы едете в автомобиле, вам не нужно думать о таких деталях, как крепежные болты, которые удерживают двигатель внутри него, или о людях, которые строят и ремонтируют дорогу, по которой он едет. Все, что вам действительно нужно знать, — это цель автомобиля (перевезти вас куда-то) и основные знания о том, как им пользоваться (например, как открывать дверь и пристегивать ремень безопасности).

Подобный уровень абстрагирования сработает, если вы просто пассажир в машине. Но если вы управляете автомобилем, вам нужно копнуть глубже и разбить свою абстракцию на несколько частей. Вы должны расширить свои знания в трех областях: сам автомобиль (например, его размер и возможности), манипуляции его частями (рулевое колесо, педаль акселератора и т. д.) и особенности дороги.

Абстрагирование от деталей может помочь при попытке найти и устранить проблемы. Предположим, что вы ведете машину и поездка дается тяжело. Вы можете

быстро оценить три основные упомянутые абстракции, связанные с автомобилем, чтобы определить источник проблемы. Если с первыми двумя абстракциями все в порядке (ваш автомобиль и способ вождения) и ни одна из них не является проблемой, можете сузить проблему до самой дороги. В таком случае выясняется, что дорога ухабистая. Теперь, если нужно, вы можете копнуть глубже в абстракцию дороги и подумать, почему ее состояние ухудшилось или, если она новая, почему строители так паршиво сделали свою работу.

Разработчики программного обеспечения используют абстракцию в качестве инструмента при создании операционной системы и ее приложений. В программном обеспечении существует множество терминов для абстрактного разделения, включая «*подсистему*», «*модуль*» и «*пакет*», но в этой главе мы будем применять термин «*компонент*», потому что так проще. При создании программного компонента разработчики обычно не задумываются о внутренней структуре других компонентов, но рассматривают те из них, которые могут задействовать (чтобы не приходилось писать дополнительное ненужное программное обеспечение), и размышляют о том, как их использовать.

В этой главе мы подробно рассмотрим компоненты, составляющие систему Linux. Хотя каждый из них имеет огромное количество технических деталей, сейчас не будем брать их в расчет и сосредоточимся на том, что именно компоненты делают по отношению ко всей системе. А детали рассмотрим в последующих главах.

1.1. Уровни абстракции в системе Linux

Использование абстракции для разделения вычислительных систем на компоненты облегчает понимание, но не работает без организации процессов. Мы объединяем компоненты в слои или уровни классификации (или группы) в соответствии с тем, где они находятся между пользователем и оборудованием. Веб-браузеры, игры и т. п. находятся на верхнем уровне, на нижнем располагается память в компьютерном оборудовании — нули и единицы. Операционная система занимает множество промежуточных уровней.

Система Linux имеет три основных уровня. На рис. 1.1 показаны эти уровни и компоненты внутри каждого из них. Аппаратное оборудование компьютера (hardware) находится на базовом уровне. Оно включает в себя память и один или несколько процессоров (CPU) для выполнения вычислений, а также для чтения из памяти и записи в нее. Такие устройства, как диски и сетевые интерфейсы, — тоже часть аппаратного оборудования.

Следующий уровень — это *ядро (kernel)*, основа операционной системы. Ядро — это программное обеспечение, содержащееся в памяти. Оно сообщает процессору, где находится его следующая задача. Выступая в качестве посредника, ядро управляет оборудованием (особенно оперативной памятью) и является основным интерфейсом между оборудованием и любой запущенной программой.



Рис. 1.1. Основные компоненты системы Linux

Процессы (processes) — запущенные программы, которыми управляет ядро, — в совокупности составляют верхний уровень системы, называемый *пользовательским пространством (user space)*. (Более конкретный термин для процесса — «*пользовательский процесс*» (*user process*), независимо от того, взаимодействует ли пользователь непосредственно с процессом. Например, все веб-серверы работают как пользовательские процессы.)

Существует огромная разница между тем, как работает ядро и пользовательские процессы: ядро работает в *режиме ядра (kernel mode)*, а пользовательские процессы — в *пользовательском режиме*. Код, работающий в режиме ядра, имеет неограниченный доступ к процессору и оперативной памяти. Такая мощная, но опасная привилегия позволяет легко повредить ядро и вывести из строя всю систему. Область памяти, доступ к которой может получить только ядро, называется *пространством ядра (kernel space)*.

Пользовательский же режим ограничен доступом к подмножеству памяти (обычно довольно небольшому) и безопасным операциям процессора. *Пользовательское пространство* — это части основной памяти, к которым могут получить доступ пользовательские процессы. Если процесс совершает ошибку и выходит из строя,

последствия локальны и устраняются с помощью ядра. Это означает, что если ваш веб-браузер выйдет из строя, он не будет продолжать научные вычисления, которые выполнялись в фоновом режиме в течение нескольких дней.

Теоретически, пользовательский процесс, вышедший из строя, не может нанести серьезного ущерба остальной части системы. На самом деле это зависит от того, что вы считаете серьезным ущербом, а также от особых привилегий процесса, ведь некоторым процессам разрешено делать больше, чем другим. Например, может ли пользовательский процесс полностью уничтожить данные на диске? Если имеет необходимые привилегии — да, и это довольно опасно. Однако существуют меры предосторожности, и большинству процессов просто не разрешается сеять хаос в системе.

ПРИМЕЧАНИЕ

Ядро Linux может запускать потоки ядра (kernel threads), очень похожие на процессы, но имеющие доступ к пространству ядра, например kthreadd и kblockd.

1.2. Оборудование: оперативная память

Оперативная память — это, пожалуй, самый важный элемент оборудования компьютерной системы. В своей первоначальной форме оперативная память — это просто огромная область хранения для группы нулей и единиц. Каждый слот для нуля или единицы называется битом. Именно в оперативной памяти хранятся запущенные ядро и процессы — по сути, тоже большие наборы битов. Все входные и выходные данные с периферийных устройств проходят через оперативную память также в виде набора битов. Процессор — это оператор памяти, он считывает свои инструкции и данные из памяти и записывает данные обратно в память.

В отношении памяти, процессов, ядра и других частей компьютерной системы часто используется термин «состояние» (*state*). Строго говоря, состояние — это особое расположение битов. Например, если у вас есть четыре бита в памяти, то биты 0110, 0001 и 1011 представляют три разных состояния системы.

Если учесть, что один процесс может легко состоять из миллионов битов в памяти, то когда речь идет о состояниях, проще использовать абстрактные термины. Вместо того чтобы описывать состояние с помощью битов, процесс описывают как заверченный или происходящий в данный момент. Например, вы можете сказать: «Процесс ожидает ввода» или «Процесс находится на втором этапе запуска».

ПРИМЕЧАНИЕ

Поскольку принято ссылаться на состояние в абстрактных терминах, а не на фактические биты, термин «образ» (*image*) относится к определенному физическому расположению битов.

1.3. Ядро

Почему мы говорим об оперативной памяти и состояниях? Почти все, что делает ядро, связано с оперативной памятью. Одной из задач ядра является разделение памяти на множество областей, и оно должно постоянно отслеживать определенную информацию об их состоянии. Каждый процесс получает некоторую часть памяти, и ядро должно обеспечивать необходимое количество памяти для каждого из процессов. Ядро отвечает за управление задачами в четырех основных областях системы.

- **Процессы.** Ядро определяет, каким процессам разрешено использовать процессор.
- **Память.** Ядро должно отслеживать распределение памяти: сколько в данный момент выделено конкретному процессу, сколько можно разделить между другими процессами и сколько свободно.
- **Драйверы устройств.** Ядро действует как интерфейс между оборудованием (например, диском) и процессами. Обычно оно управляет подключенным оборудованием.
- **Системные вызовы и поддержка.** Процессы обычно используют системные вызовы для связи с ядром.

Далее мы кратко изучим каждую из этих областей.

ПРИМЕЧАНИЕ

Если вы хотите изучить работу ядра более подробно, то я советую прочитать две отличные книги на эту тему: десятое издание книги Ави Зильбершаца *Operating System Concepts* (Wiley, 2018) и четвертое издание книги Эндрю Таненбаума *Modern Operating Systems* (Prentice Hall, 2014).

1.3.1. Управление процессами

Управление процессами описывает запуск, приостановку, возобновление, планирование и завершение процессов. Концепции, лежащие в основе запуска и завершения процессов, довольно просты, но описание того, как процессом используется процессор, немного сложнее.

В любой современной операционной системе многие процессы выполняются параллельно. Например, у вас могут быть открыты веб-браузер и электронная таблица одновременно. Однако все не так, как кажется: процессы, стоящие за этими приложениями, обычно не выполняются в одно и то же время.

Рассмотрим систему с одноядерным процессором. Многие процессы могут задействовать процессор, но только один из них фактически использует процессор

в любой момент времени. На практике процесс работает с процессором в течение небольшой доли секунды и делает паузу, затем другой процесс занимает процессор в течение еще одной небольшой доли секунды, потом в дело вступает еще один процесс и т. д. Акт передачи одним процессом управления процессором другому процессу называется *переключением контекста*.

Каждый отрезок времени, называемый *квантом времени*, позволяет процессу выполнить значительные вычисления (и действительно, процесс часто завершает свою задачу в течение одного кванта времени). Однако из-за того, что кванты очень малы, пользователи их не воспринимают, и возникает ощущение, что система выполняет несколько процессов одновременно (режим многозадачности).

Ядро отвечает за переключение контекста. Чтобы понять, как это работает, рассмотрим ситуацию, в которой процесс работает в пользовательском режиме, но его временной квант истек. Вот что происходит.

1. Процессор (фактическое оборудование) прерывает текущий процесс на основе внутреннего таймера, переключается в режим ядра и передает управление обратно ядру.
2. Ядро записывает текущее состояние процессора и памяти, что необходимо для возобновления только что прерванного процесса.
3. Ядро выполняет любые задачи, которые могли возникнуть в течение предыдущего временного кванта (например, сбор данных из ввода-вывода).
4. Теперь ядро готово к запуску другого процесса. Оно анализирует список процессов, готовых к запуску, и выбирает один из них.
5. Ядро подготавливает память для нового процесса, а затем готовит к нему процессор.
6. Ядро сообщает процессору длительность временного кванта для нового процесса.
7. Ядро переключает процессор в пользовательский режим и передает управление процессором процессу.

Переключение контекста позволяет понять, когда именно запускается ядро. Суть заключается в том, что ядро запускается между временными квантами процесса во время переключения контекста.

В случае многопроцессорной системы, как и в большинстве современных машин, все немного сложнее, потому что ядро не перестает управлять текущим процессором, чтобы позволить процессу работать на другом процессоре, и одновременно могут выполняться несколько процессов. Но чтобы максимально задействовать все доступные процессоры, ядро в любом случае выполняет необходимые шаги (и может использовать определенные лазейки, чтобы занять больше времени процессора для себя).

1.3.2. Управление памятью

Ядро должно управлять памятью во время переключения контекста, а это довольно сложная задача. Должны выполняться следующие условия.

- Ядро должно иметь в памяти выделенную область, к которой пользовательские процессы не могут получить доступ.
- Каждому пользовательскому процессу необходима собственная область памяти.
- Один пользовательский процесс не может получить доступ к области памяти, выделенной другому процессу.
- Пользовательские процессы могут совместно работать с памятью.
- Часть памяти пользовательских процессов может быть доступна только для чтения.
- Система может использовать больше памяти, чем ее существует физически, задействуя дисковое пространство в качестве вспомогательного механизма.

К счастью для ядра, оно не одно выполняет всю работу. Современные процессоры включают в себя блок управления памятью (memory management unit, ММУ), который обеспечивает схему доступа к памяти, называемую *виртуальной памятью*. При использовании виртуальной памяти процесс не получает прямого доступа к памяти через ее физическое местоположение в компьютерной системе. Вместо этого ядро настраивает каждый процесс, чтобы он действовал так, будто ему доступна вся система. Когда процесс обращается к части своей памяти, ММУ перехватывает обращение и с помощью таблицы соответствий преобразует адрес памяти с точки зрения процесса в фактическое физическое местоположение памяти в системе. Ядро по-прежнему должно инициализировать, постоянно поддерживать и изменять таблицу соответствий адресов памяти. Например, во время переключения контекста ядро должно заменить таблицу соответствий исходного процесса на таблицу последующего процесса.

ПРИМЕЧАНИЕ

Реализация таблицы соответствий адресов памяти называется таблицей страниц.

Подробнее о том, как отслеживать производительность памяти, описано в главе 8.

1.3.3. Управления драйверами устройств

Роль ядра в работе с устройствами относительно проста. Устройство обычно доступно только в режиме ядра, поскольку неправильный доступ (например, пользовательский процесс, запрашивающий отключение питания) может привести к сбою системы. Значительная проблема заключается в том, что различные устройства редко имеют один и тот же интерфейс программирования, даже если устройства